Own Project - Stroke Prediction

2022-10-25

Introduction

In this project, clinical patient data are analyzed to predict a stroke event in a patient. The analyzed dataset contains patient data listed with some characteristics. These characteristics are: gender, age, hypertension, heart_disease, ever_married, work_type, Residence_type, avg_glucose_level, bmi, smoking_status, stroke. The dataset contains a little more than 5000 records.

Since strokes are responsible for many deaths each year, it is important to identify the factors that play a role and predict whether a patient will have a stroke based on the degree to which these factors are present in the patient.

In the first step, the data are viewed and cleaned for further analysis. In the second step, the data are analyzed and the individual characteristics are analyzed in more detail and correlations are searched for. In the final step, three models are applied to the data to predict stroke. In this step, the models are experimented with to achieve the best results for prediction. The models applied are Logistic Regression, Random Forest and XGBoost.

Packages

It is checked if all required packages are installed, if not required packages will be installed

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.8     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: caret
## Lade nötiges Paket: lattice
##
## Attache Paket: 'caret'
##
## Das folgende Objekt ist maskiert 'package:purrr':
##
##     lift
```

```
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: data.table
##
## Attache Paket: 'data.table'
##
## Die folgenden Objekte sind maskiert von 'package:dplyr':
##
##     between, first, last
##
## Das folgende Objekt ist maskiert 'package:purrr':
##
##     transpose
```

```
if(!require(naniar)) install.packages("naniar", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: naniar
```

```
if(!require(caret)) install.packages("carnet", repos = "http://cran.us.r-project.org")
if(!require(caTools)) install.packages("caTools", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: caTools
```

```
if(!require(corrplot)) install.packages("corrplot", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: corrplot
## corrplot 0.92 loaded
```

```
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: randomForest
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attache Paket: 'randomForest'
##
## Das folgende Objekt ist maskiert 'package:dplyr':
##
##     combine
##
## Das folgende Objekt ist maskiert 'package:ggplot2':
##
##     margin
```

```
if(!require(imbalance)) install.packages("imbalance", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: imbalance
```

```
if(!require(MASS)) install.packages("MASS", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: MASS
##
## Attache Paket: 'MASS'
##
## Das folgende Objekt ist maskiert 'package:dplyr':
##
##     select
```

```
if(!require(ROSE)) install.packages("ROSE", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: ROSE
## Loaded ROSE 0.0-4
```

```
if(!require(broom)) install.packages("broom", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: broom
```

```
if(!require(margins)) install.packages("margins", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: margins
```

```
if(!require(yardstick)) install.packages("yardstick", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: yardstick
## For binary classification, the first factor level is assumed to be the event.
## Use the argument `event_level = "second"` to alter this as needed.
##
## Attache Paket: 'yardstick'
##
## Die folgenden Objekte sind maskiert von 'package:caret':
##
##     precision, recall, sensitivity, specificity
##
## Das folgende Objekt ist maskiert 'package:readr':
##
##     spec
```

```
if(!require(ROCR)) install.packages("ROCR", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: ROCR
##
## Attache Paket: 'ROCR'
##
## Das folgende Objekt ist maskiert 'package:margins':
##
##     prediction
```

```
if(!require(glmnet)) install.packages("glmnet", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: glmnet
## Lade nötiges Paket: Matrix
##
## Attache Paket: 'Matrix'
##
## Die folgenden Objekte sind maskiert von 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-4
```

```
if(!require(ranger)) install.packages("ranger", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: ranger
##
## Attache Paket: 'ranger'
##
## Das folgende Objekt ist maskiert 'package:randomForest':
##
##     importance
```

```
if(!require(evaluate)) install.packages("evaluate", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: evaluate
```

```
if(!require(kernlab)) install.packages("kernlab", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: kernlab
##
## Attache Paket: 'kernlab'
##
## Das folgende Objekt ist maskiert 'package:purrr':
##
##     cross
##
## Das folgende Objekt ist maskiert 'package:ggplot2':
##
##     alpha
```

```
if(!require(xgboost)) install.packages("xgboost", repos = "http://cran.us.r-project.org")
```

```
## Lade nötiges Paket: xgboost
##
## Attache Paket: 'xgboost'
##
## Das folgende Objekt ist maskiert 'package:dplyr':
##
##     slice
```

Packets are loaded

```
library(tidyverse)
library(caret)
library(data.table)
library(naniar)
library(caret)
library(caTools)
library(corrplot)
library(randomForest)
library(imbalance)
library(MASS)
library(ROSE)
library(broom)
library(margins)
library(yardstick)
library(ROCR)
library(glmnet)
library(ranger)
library(evaluate)
library(kernlab)
library(xgboost)
```

Data Import

The Dataset is online available to download: https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset?datasetId=1120859&language=R (https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset?datasetId=1120859&language=R)

The dataset is stored in github and is loaded directly from there

```
data <- read.csv("https://raw.githubusercontent.com/Flow2992/DataScience/main/healthcare-dataset-stroke-data.csv") # Load Dataset from github
data <- as.data.table(data)
```

Data First look and data cleaning

Data structure

```
str(data)
```

```
## Classes 'data.table' and 'data.frame':   5110 obs. of  12 variables:
##  $ id               : int  9046 51676 31112 60182 1665 56669 53882 10434 27419 60491 ...
##  $ gender           : chr  "Male" "Female" "Male" "Female" ...
##  $ age              : num  67 61 80 49 79 81 74 69 59 78 ...
##  $ hypertension     : int  0 0 0 0 1 0 1 0 0 0 ...
##  $ heart_disease    : int  1 0 1 0 0 0 1 0 0 0 ...
##  $ ever_married     : chr  "Yes" "Yes" "Yes" "Yes" ...
##  $ work_type        : chr  "Private" "Self-employed" "Private" "Private" ...
##  $ Residence_type   : chr  "Urban" "Rural" "Rural" "Urban" ...
##  $ avg_glucose_level: num  229 202 106 171 174 ...
##  $ bmi              : chr  "36.6" "N/A" "32.5" "34.4" ...
##  $ smoking_status   : chr  "formerly smoked" "never smoked" "never smoked" "smokes" ...
##  $ stroke           : int  1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
summary (data)
```

```
##        id           gender               age         hypertension
##  Min.   :   67   Length:5110        Min.   : 0.08   Min.   :0.00000
##  1st Qu.:17741   Class :character   1st Qu.:25.00   1st Qu.:0.00000
##  Median :36932   Mode  :character   Median :45.00   Median :0.00000
##  Mean   :36518                      Mean   :43.23   Mean   :0.09746
##  3rd Qu.:54682                      3rd Qu.:61.00   3rd Qu.:0.00000
##  Max.   :72940                      Max.   :82.00   Max.   :1.00000
##  heart_disease     ever_married        work_type         Residence_type
##  Min.   :0.00000   Length:5110        Length:5110        Length:5110
##  1st Qu.:0.00000   Class :character   Class :character   Class :character
##  Median :0.00000   Mode  :character   Mode  :character   Mode  :character
##  Mean   :0.05401
##  3rd Qu.:0.00000
##  Max.   :1.00000
##  avg_glucose_level     bmi            smoking_status         stroke
##  Min.   : 55.12    Length:5110        Length:5110        Min.   :0.00000
##  1st Qu.: 77.25    Class :character   Class :character   1st Qu.:0.00000
##  Median : 91.89    Mode  :character   Mode  :character   Median :0.00000
##  Mean   :106.15                                          Mean   :0.04873
##  3rd Qu.:114.09                                          3rd Qu.:0.00000
##  Max.   :271.74                                          Max.   :1.00000
```

The ID column is not needed and therefore deleted

```
data = subset(data, select = -c(id))
```

Check if data is missing in a variable

```
miss_scan_count(data = data, search = list("N/A", "Unknown")) # Data are missing in the varia
bles "bmi" and "smoking_status"
```

```
## # A tibble: 11 x 2
##    Variable              n
##    <chr>             <int>
##  1 gender                0
##  2 age                   0
##  3 hypertension          0
##  4 heart_disease         0
##  5 ever_married          0
##  6 work_type             0
##  7 Residence_type        0
##  8 avg_glucose_level     0
##  9 bmi                 201
## 10 smoking_status     1544
## 11 stroke                0
```

Check which values the individual variables have. The variables “age”, “avg_glucose_level” and “bmi” are not checked, because they contain to many different values

```
table(data$gender)
```

```
##
## Female   Male  Other
##   2994   2115      1
```

```
table(data$hypertension)
```

```
##
##    0    1
## 4612  498
```

```
table(data$heart_disease)
```

```
##
##    0    1
## 4834  276
```

```
table(data$ever_married)
```

```
##
##   No  Yes
## 1757 3353
```

```
table(data$work_type)
```

```
##
##      children      Govt_job  Never_worked       Private Self-employed
##           687           657            22          2925           819
```

```
table(data$Residence_type)
```

```
##
## Rural Urban
##  2514  2596
```

```
table(data$smoking_status)
```

```
##
## formerly smoked    never smoked          smokes         Unknown
##             885            1892             789            1544
```

```
table(data$stroke)
```

```
##
##    0    1
## 4861  249
```

As data shows there is one record within the variable gender with the value “other”. This will be removed

```
data <- data %>% filter(data$gender!='Other')
table(data$gender)
```

```
##
## Female   Male
##   2994   2115
```

After the individual variables have been checked, the first task is to edit the missing bmi data. The missing values will be replaced with mean values by gender. This approach was chosen because the average BMI of men is higher than that of women.

Replacing missing BMI values

```
suppressWarnings(data$bmi <- as.numeric(as.character(data$bmi))) # transform bmi column to nu
meric, because the missing values werw not numeric

gender_mean_bmi <- data %>% group_by(gender) %>% summarise(bmi = mean(bmi, na.rm = TRUE)) # c
alculate gender bmi
gender_mean_bmi
```

```
## # A tibble: 2 x 2
##   gender   bmi
##   <chr>  <dbl>
## 1 Female  29.1
## 2 Male    28.6
```

```
data[gender == 'Female' & is.na(data$bmi), bmi := gender_mean_bmi[1, 'bmi']] # replace missin
g female bmi values calculated mean bmi
data[gender == 'Male'   & is.na(data$bmi), bmi := gender_mean_bmi[2, 'bmi']] # replace missin
g male bmi values calculated mean bmi

# Recheck for missing data
miss_scan_count(data = data, search = list("N/A", "Unknown"))
```

```
## # A tibble: 11 x 2
##    Variable              n
##    <chr>             <int>
##  1 gender                0
##  2 age                   0
##  3 hypertension          0
##  4 heart_disease         0
##  5 ever_married          0
##  6 work_type             0
##  7 Residence_type        0
##  8 avg_glucose_level     0
##  9 bmi                   0
## 10 smoking_status     1544
## 11 stroke                0
```

The second task is to edit the smoking_status “Unknown”. This status is not suitable for further analysis, therefore the unknown values are replaced. In order not to change the data too much, the status “Unknown” is replaced according to the distribution of the other values. The distribution of the other statuses is calculated and the status “Unknown” is replaced according to the distribution.

Calculation of the probabilities

```
table(data$smoking_status) # smoking status contains a lot of Values "Unknown".
```

```
##
## formerly smoked    never smoked          smokes         Unknown
##             884            1892             789            1544
```

```
s_dist1 <- ggplot(data, aes(x = smoking_status)) + geom_bar()  # safe plot distribution befor
e replacing the status "Unknown"

FS <- 885 / (3566) # Calculate probability for status formerly smoked
NS <- 1892 / (3566) # Calculate probability for status never smoked
S <- 789 / (3566) # Calculate probability for status smokes
```

Replace missing values on the basis of the calculated probabilities and remove supporting columns

```
data$prob <- runif(nrow(data))
data <- data %>% mutate(Proba = ifelse(prob <= FS, "formerly smoked", ifelse(prob <= (FS+NS),
"never smoked", ifelse(prob <= 1, "smokes", "Check"))))
data <- data %>% mutate(smoking_status = ifelse(smoking_status == "Unknown", Proba, smoking_s
tatus))

table(data$smoking_status) # ReCheck smoking values distibution
```

```
##
## formerly smoked    never smoked          smokes
##            1256            2721            1132
```

```
# Delete supporting columns for replacing the status
data = subset(data, select = -c(prob))
data = subset(data, select = -c(Proba))

miss_scan_count(data = data, search = list("N/A", "Unknown")) # recheck dataset for missing v
alues
```

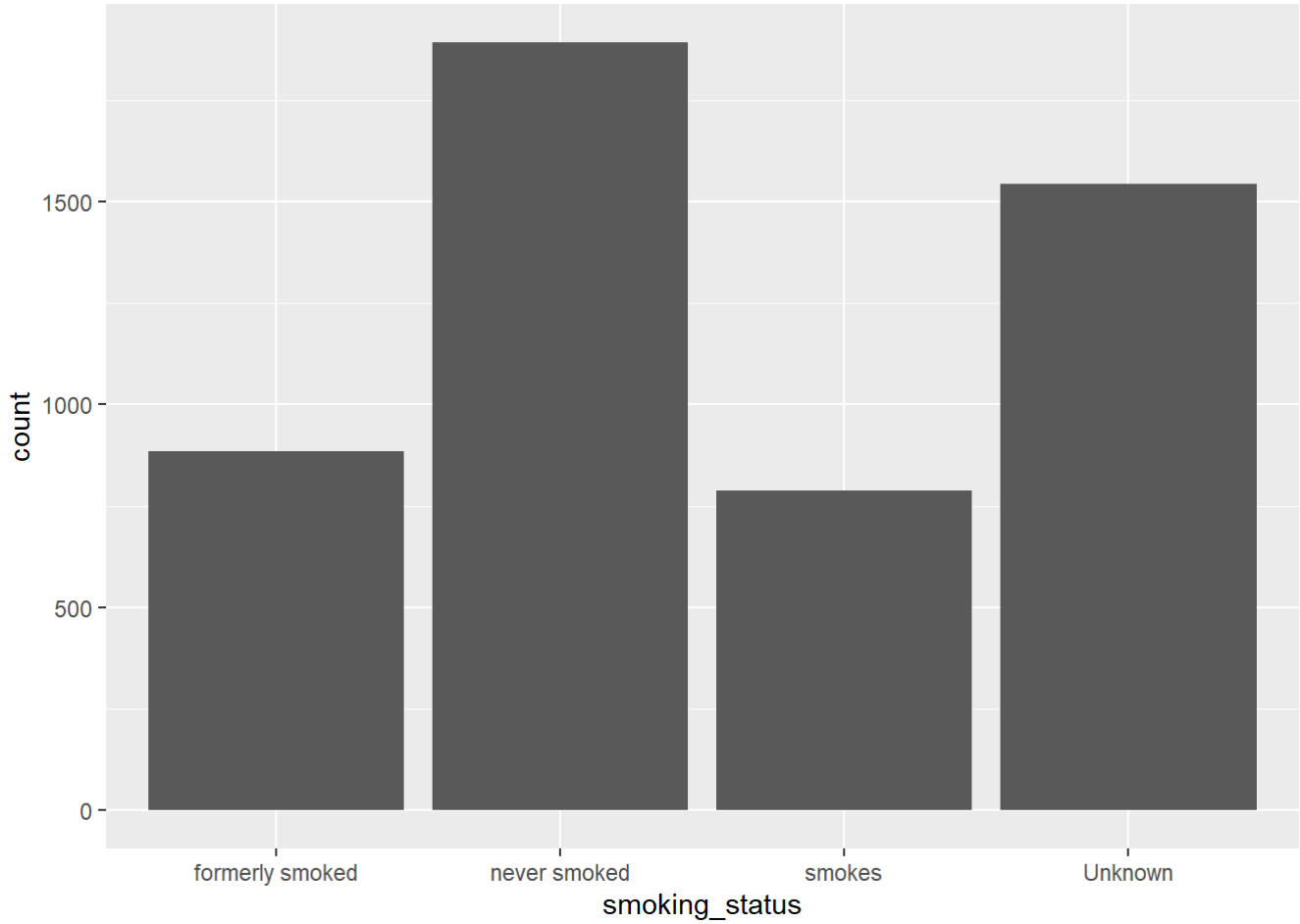
```
## # A tibble: 11 x 2
##    Variable              n
##    <chr>             <int>
##  1 gender                0
##  2 age                   0
##  3 hypertension          0
##  4 heart_disease         0
##  5 ever_married          0
##  6 work_type             0
##  7 Residence_type        0
##  8 avg_glucose_level     0
##  9 bmi                   0
## 10 smoking_status        0
## 11 stroke                0
```

```
s_dist2 <- ggplot(data, aes(x = smoking_status)) + geom_bar() # safe plot distibution after r
eplacing the status "Unknown"
```

Visual Comparison of Distibution of smoking values before and after replacing “Unknown” values

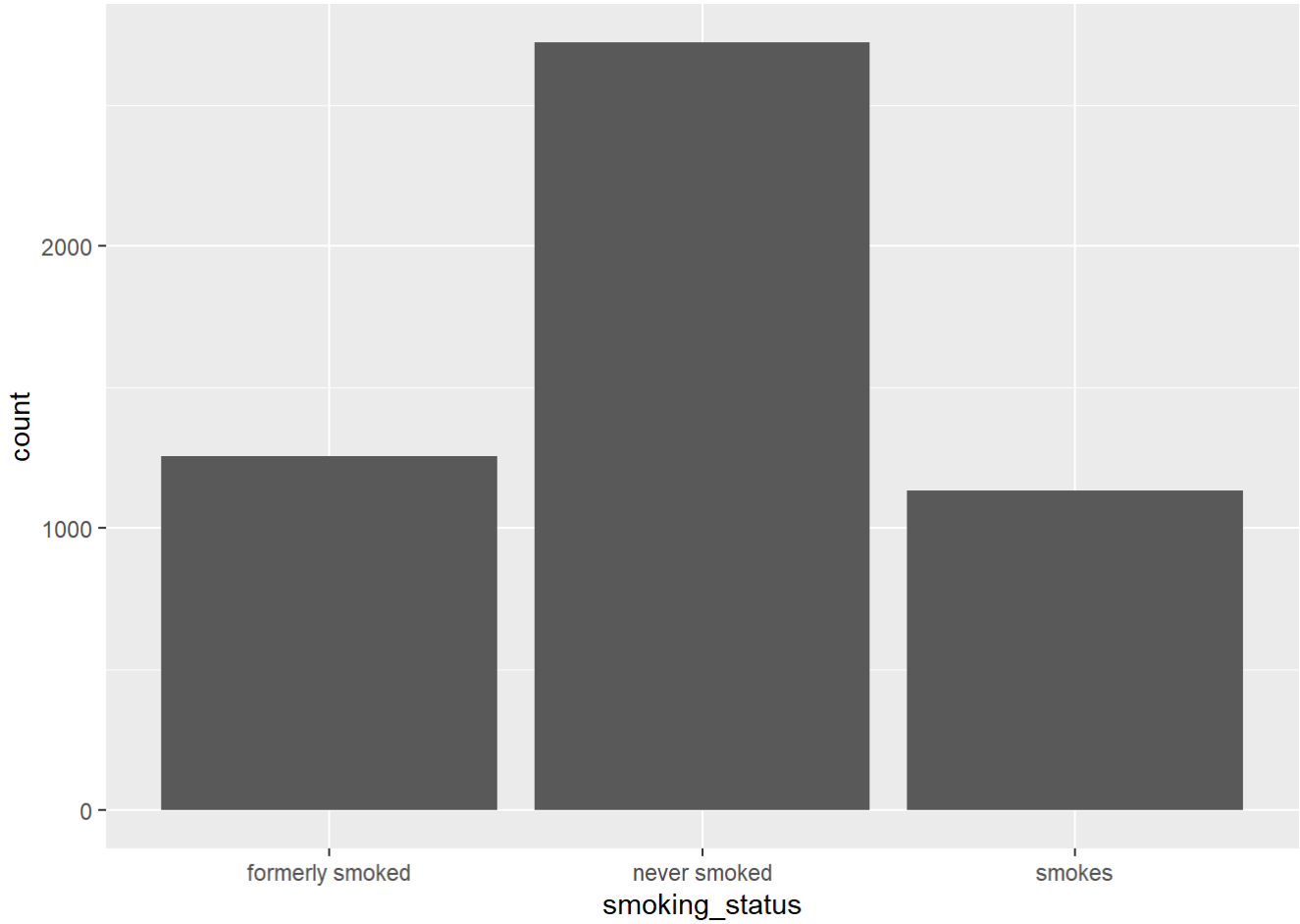
Before replacing the “Unknown” status

```
s_dist1
```



After replacing the “Unknown” status

```
s_dist2
```

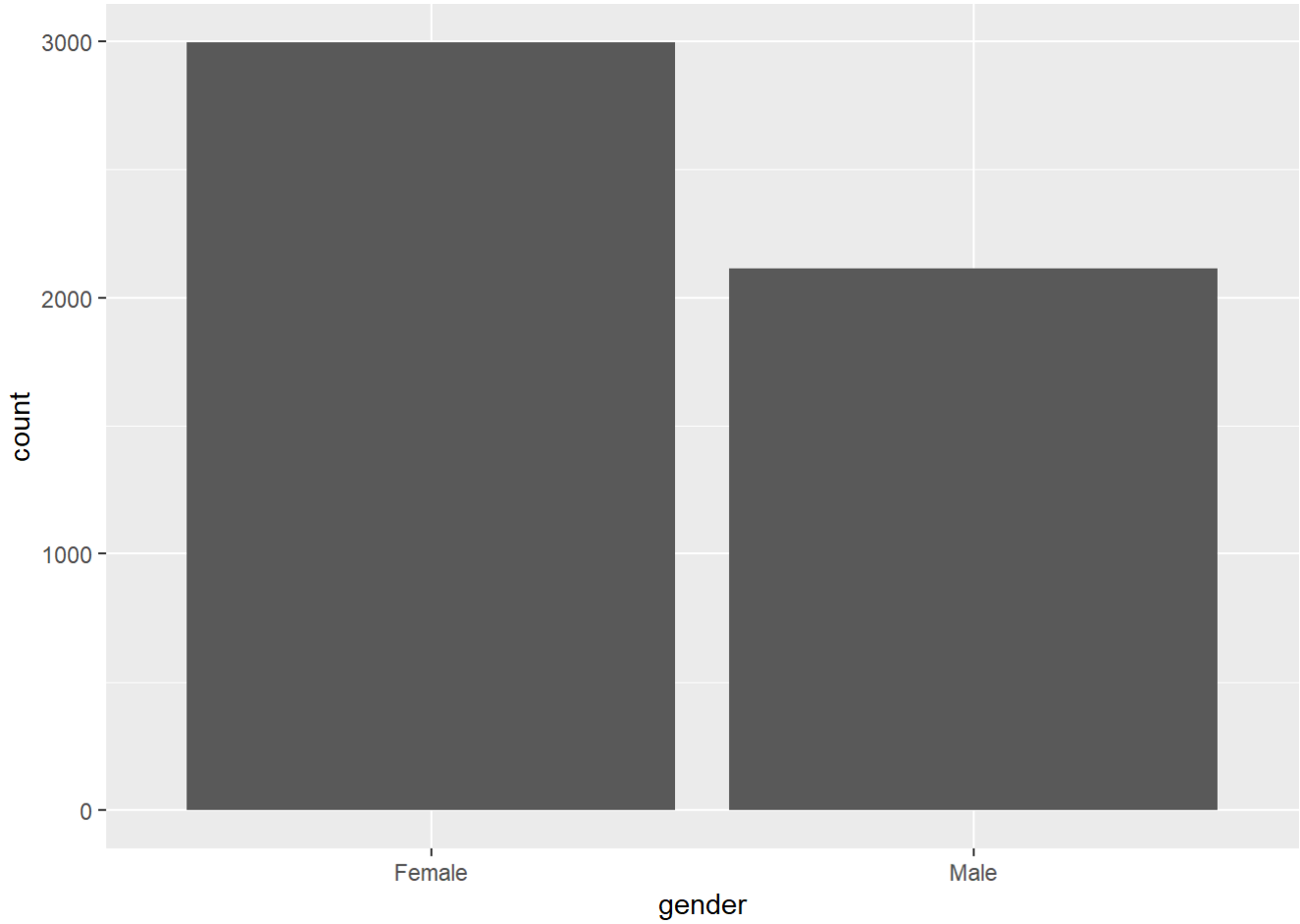


Overall distibution after replacing the “Unknown” status didn’t change much, so the replacment worked well

Data exploration

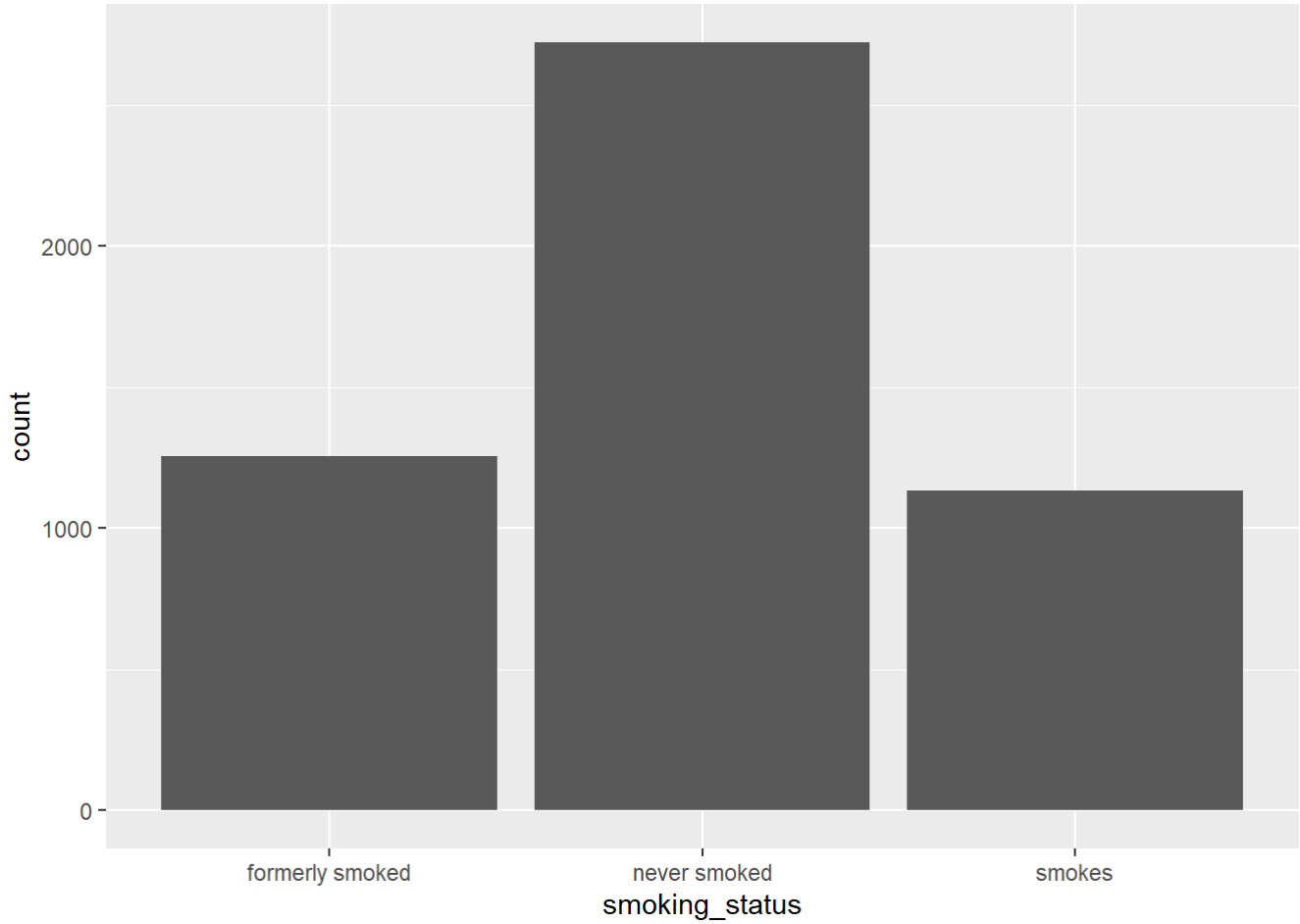
Gender distribution

```
ggplot(data, aes(x = gender)) + geom_bar()
```



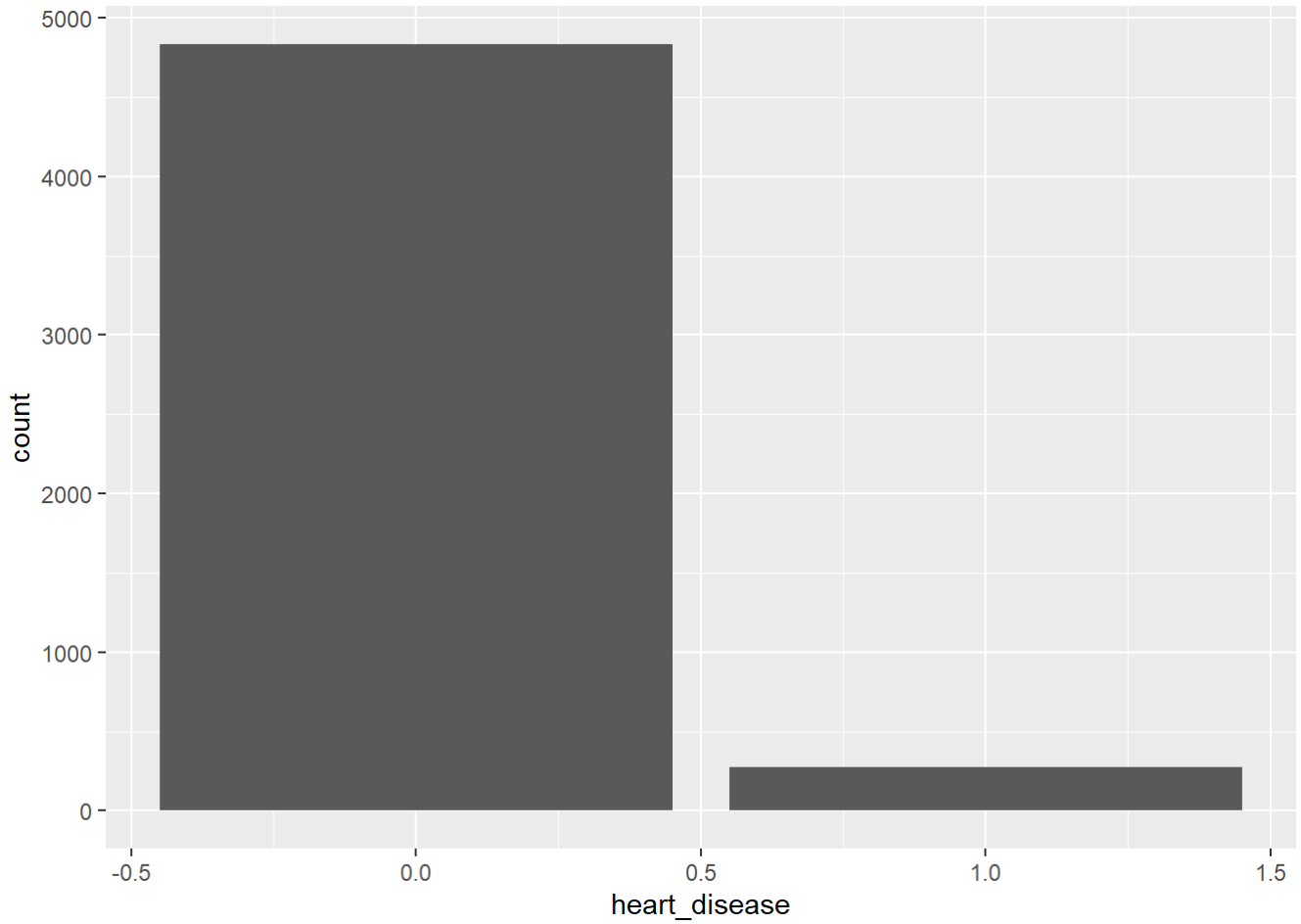
Smoking_status distribution

```
ggplot(data, aes(x = smoking_status)) + geom_bar()
```



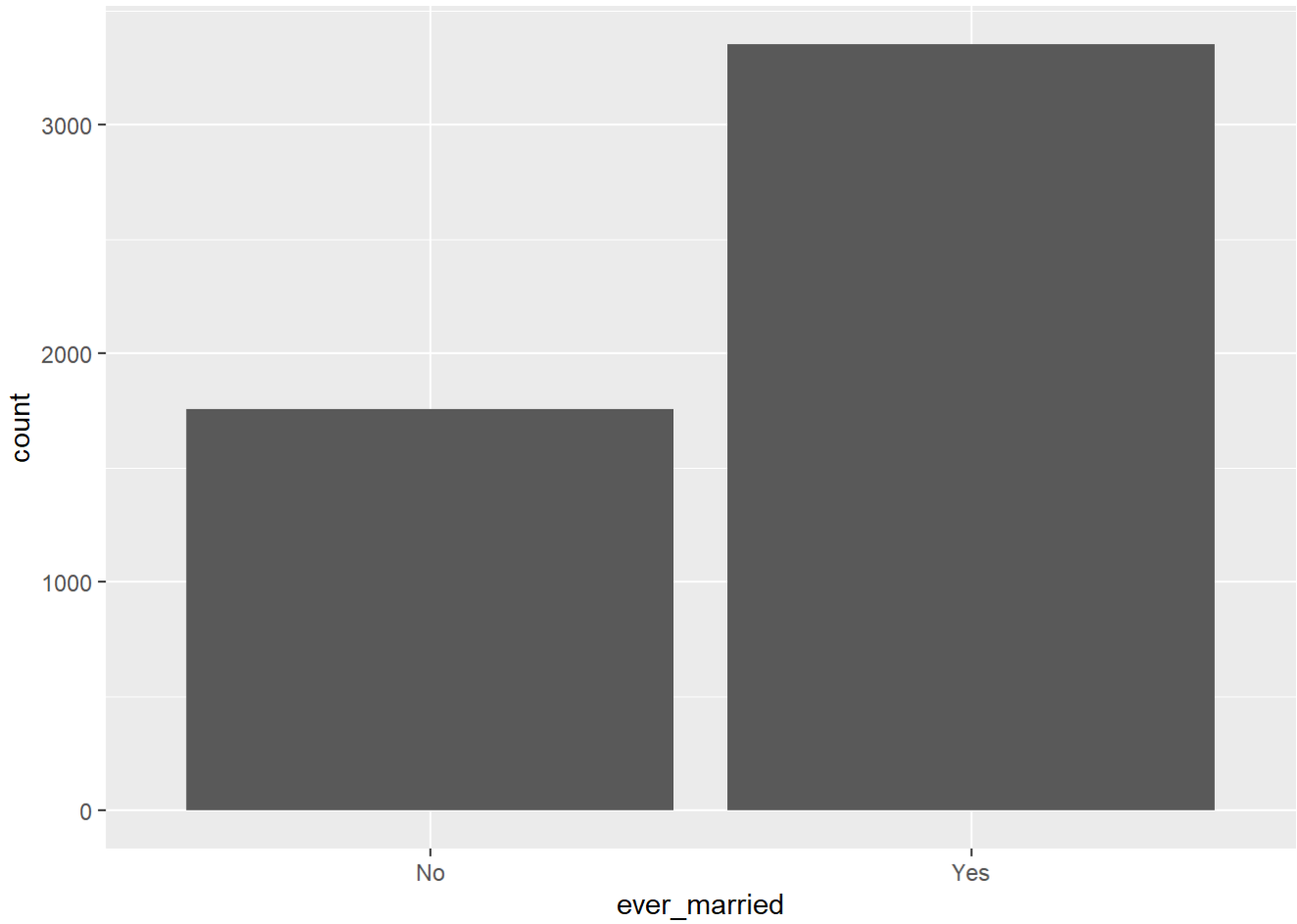
Heart_disease distribution

```
ggplot(data, aes(x = heart_disease)) + geom_bar()
```



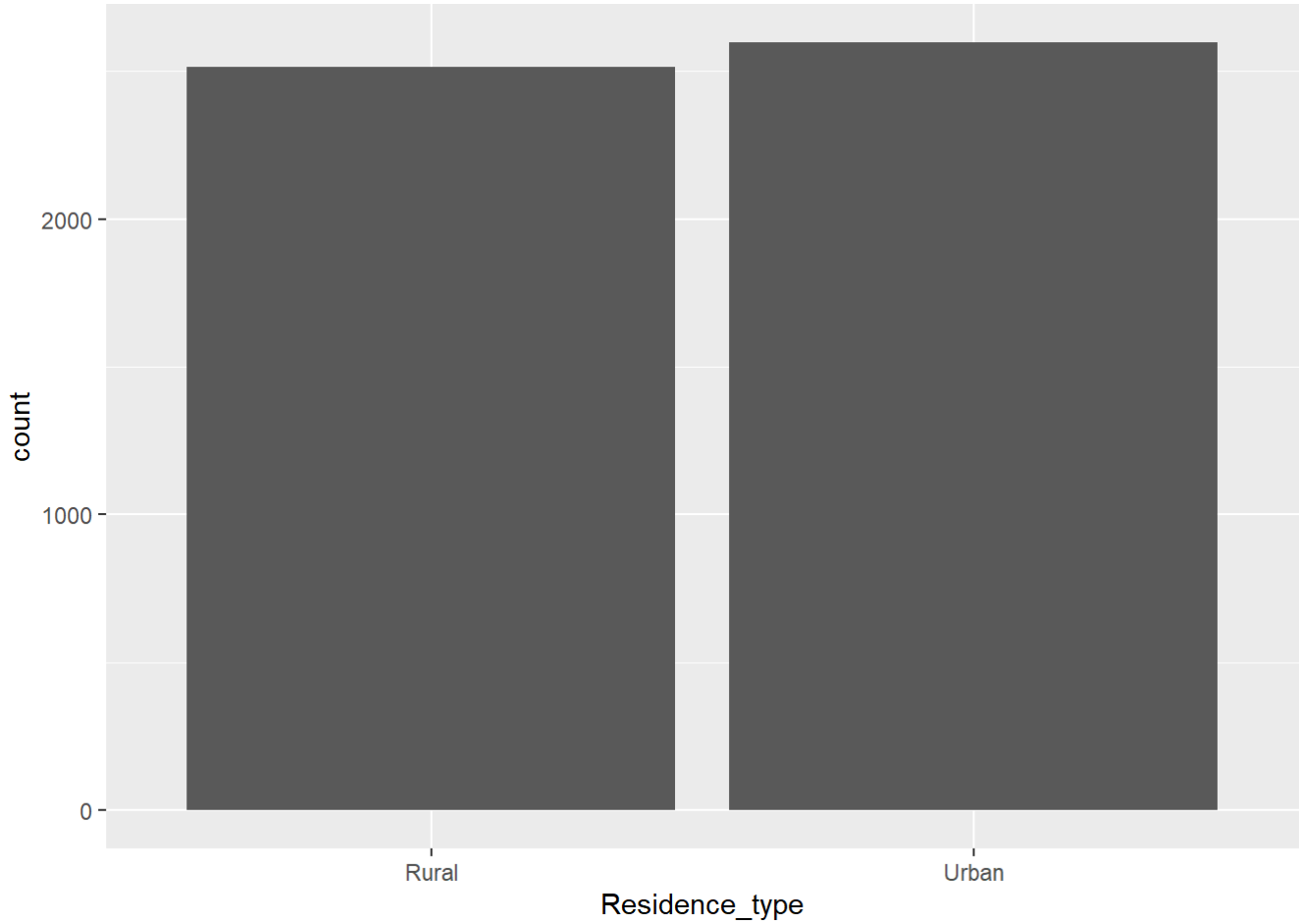
Ever_married distribution

```
ggplot(data, aes(x = ever_married)) + geom_bar()
```



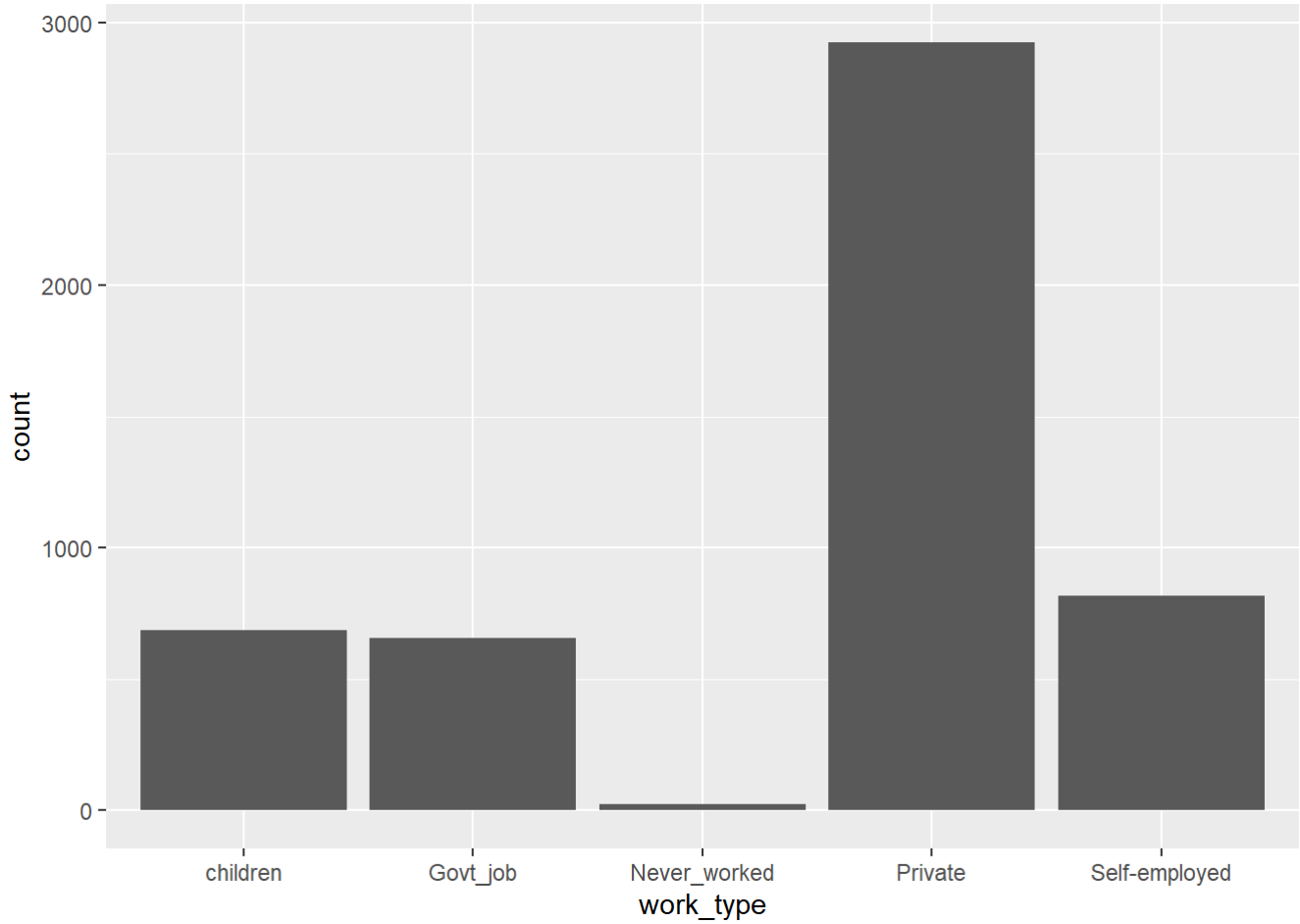
Residence_type distribution

```
ggplot(data, aes(x = Residence_type)) + geom_bar()
```



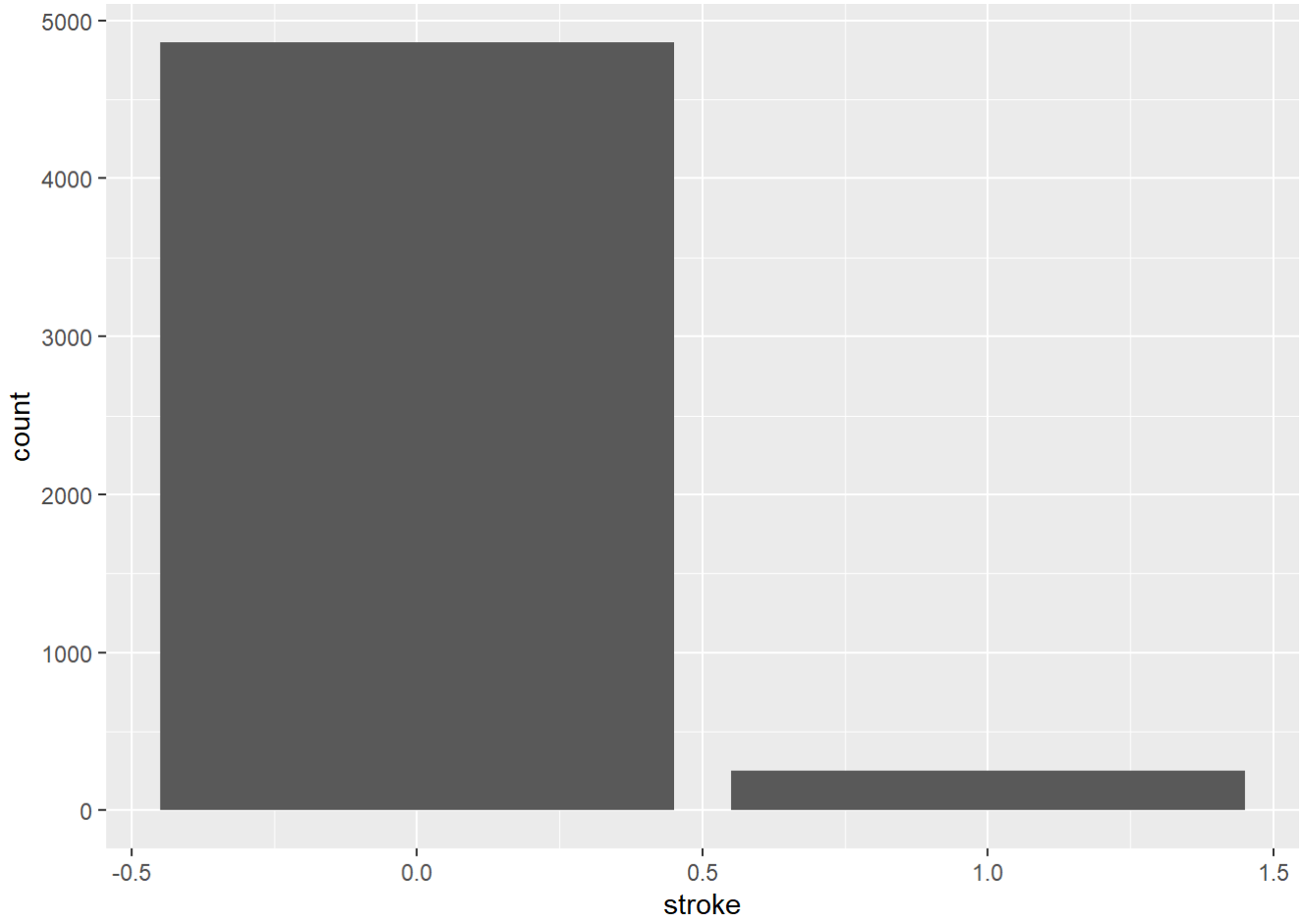
Work_type distribution

```
ggplot(data, aes(x = work_type)) + geom_bar()
```



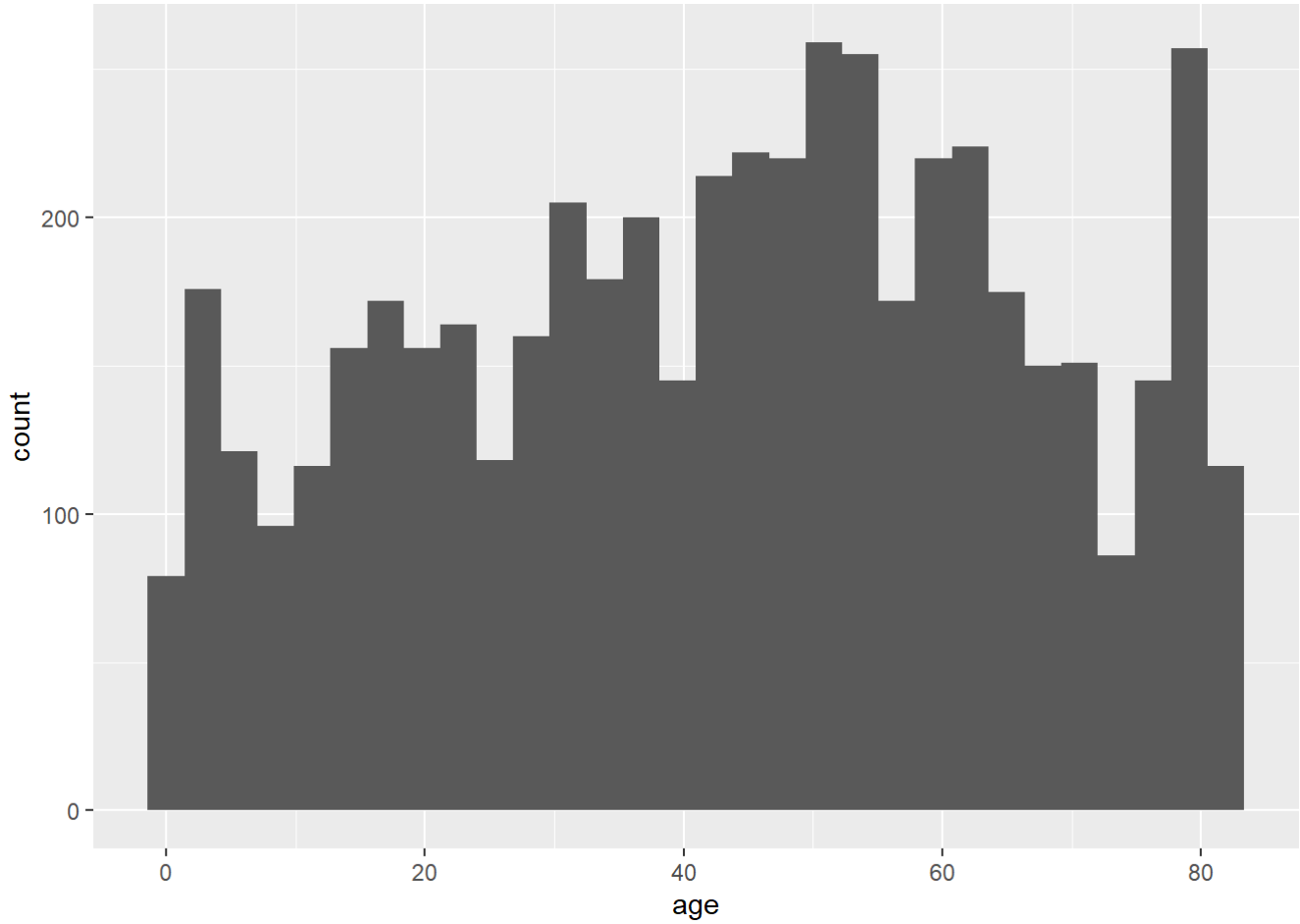
Stroke distribution

```
ggplot(data, aes(x = stroke)) + geom_bar()
```



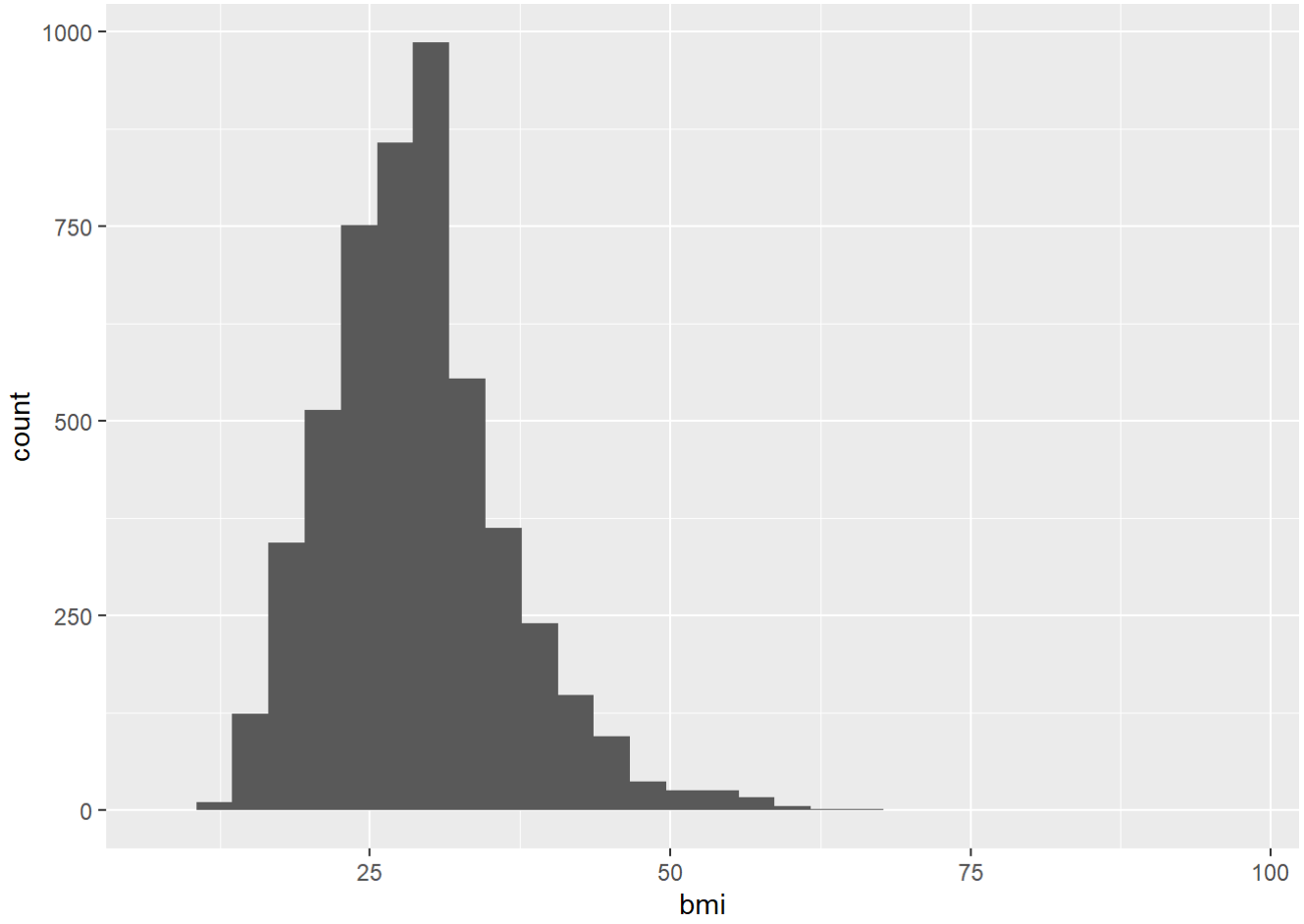
Age distribution

```
ggplot(data, aes(x = age)) + geom_histogram(bins=30)
```



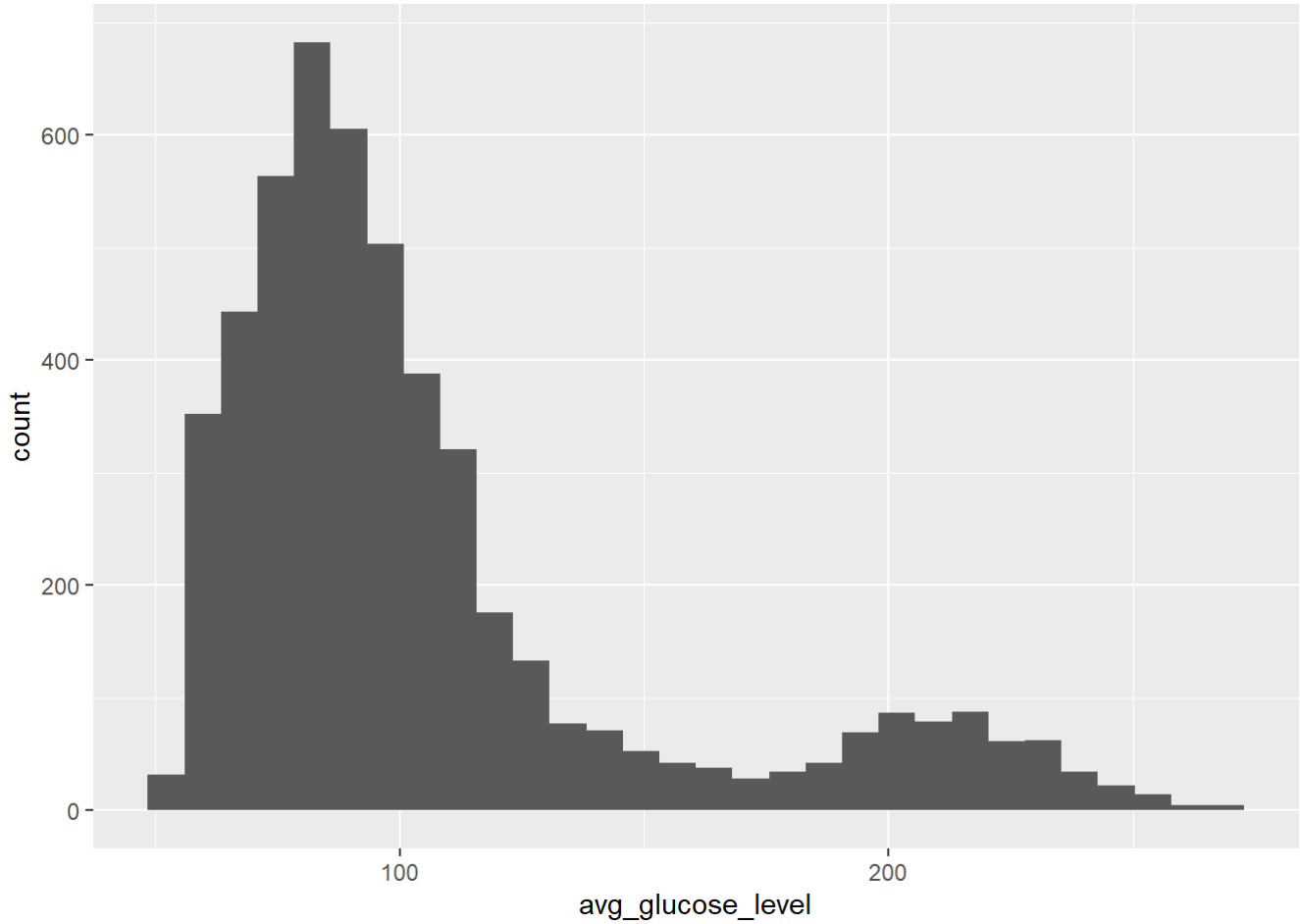
Bmi distribution

```
ggplot(data, aes(x = bmi)) + geom_histogram(bins=30)
```



Avg_glucose_level distribution

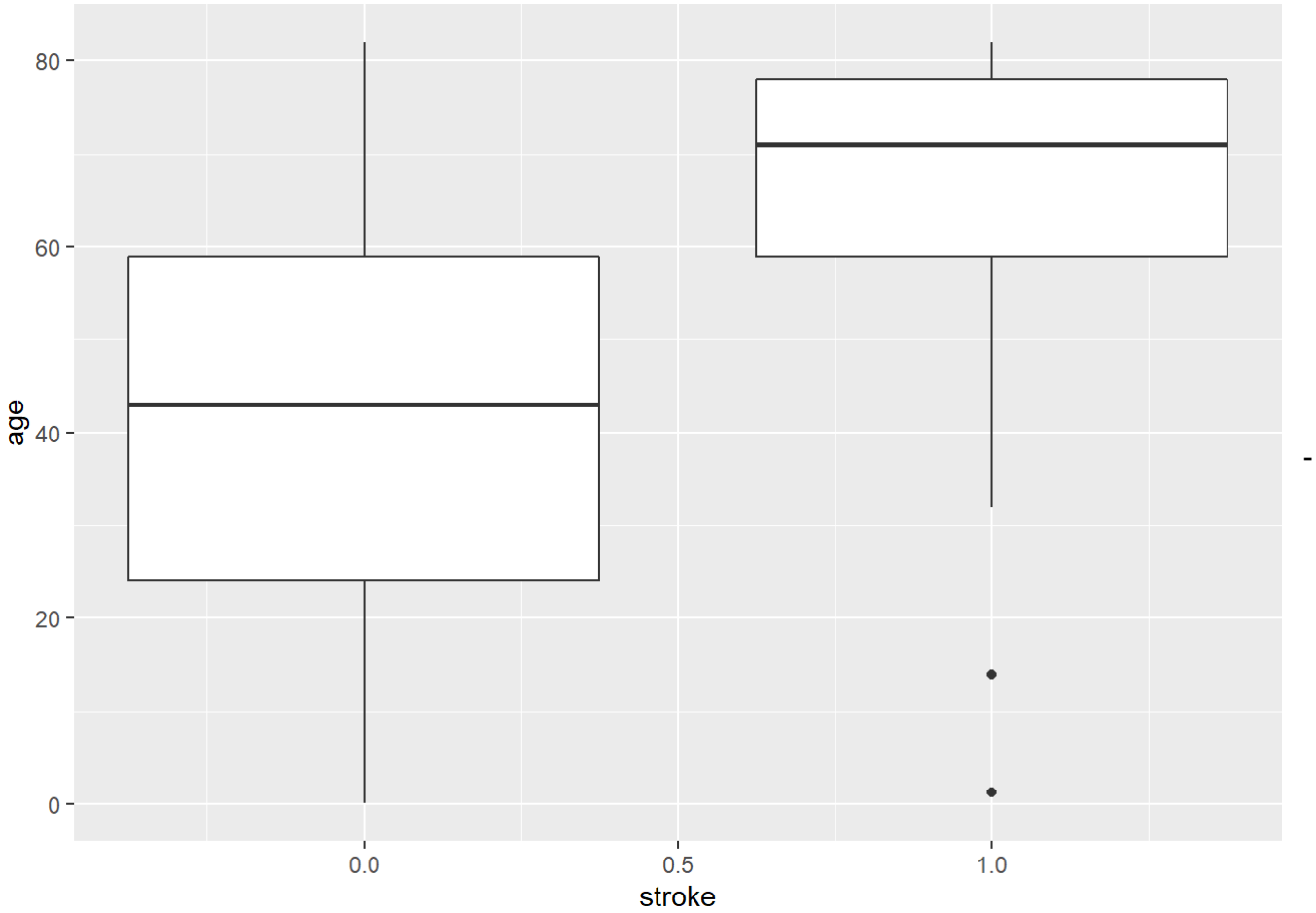
```
ggplot(data, aes(x = avg_glucose_level)) + geom_histogram(bins=30)
```



Visual inspection of the data distribution of variables with many expressions in combination with the stroke data to learn more about the data and potential correlations.

Combination of age and stroke

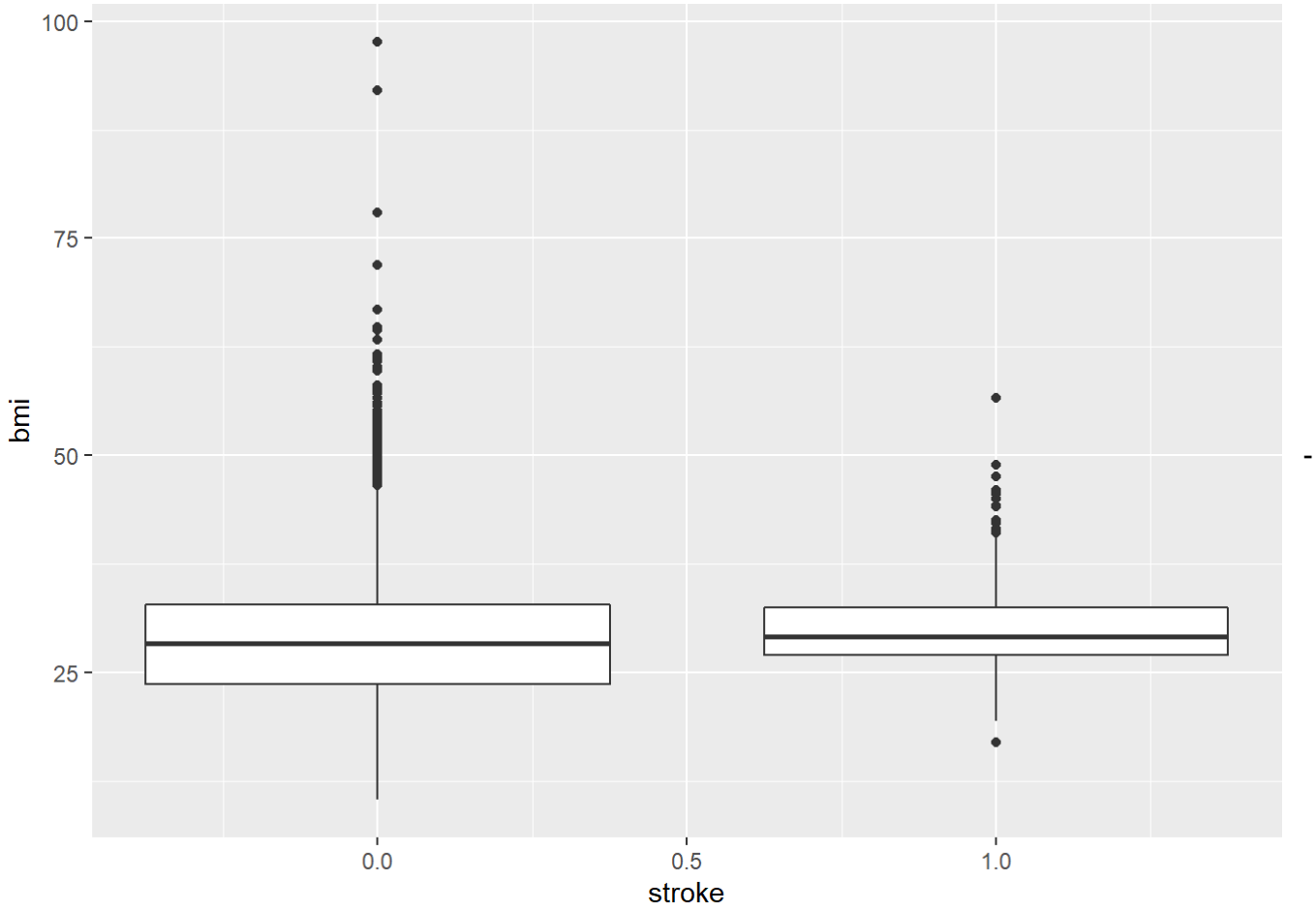
```
ggplot(data, aes(x = stroke, y = age, group = stroke)) +geom_boxplot()
```



> age seems to have some sort of impact on stroke

Combination of bmi and stroke

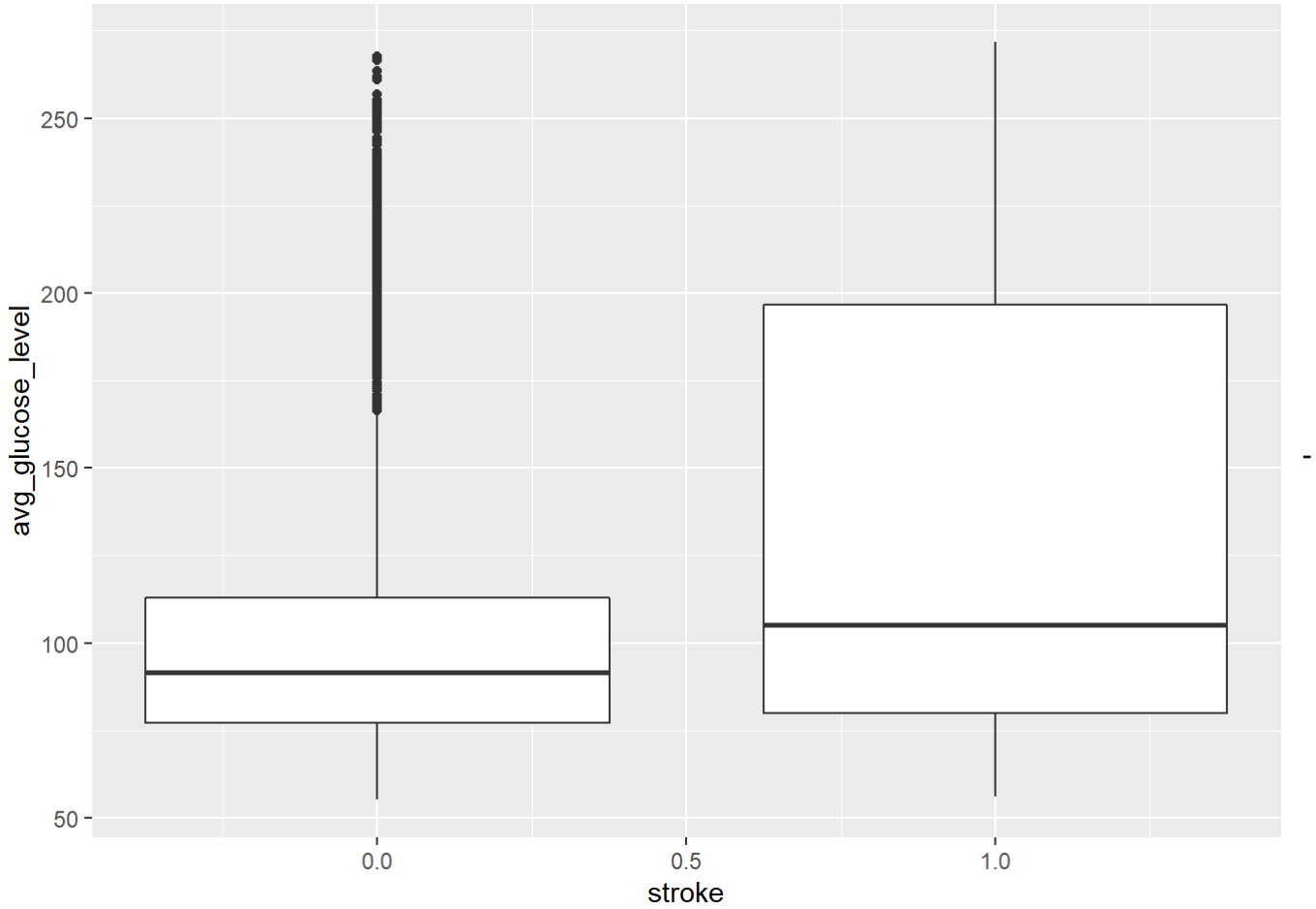
```
ggplot(data, aes(x = stroke, y = bmi, group = stroke)) +geom_boxplot()
```



> no clear visual correlation

Combination of avg_glucose_level and stroke

```
ggplot(data, aes(x = stroke, y = avg_glucose_level, group = stroke)) +geom_boxplot()
```



> correlation possible

For further analysis, some text values of the variables have to be converted into numerical values

```
# gender
data$gender[data$gender == "Male"] <- 1 # Male --> 1
data$gender[data$gender == "Female"] <- 0 # Female --> 0

# residence typ
data$Residence_type[data$Residence_type == "Urban"] <- 1 # Urlan --> 1
data$Residence_type[data$Residence_type == "Rural"] <- 0 # Rural --> 0

# ever married
data$ever_married[data$ever_married == "Yes"] <- 1 # yes --> 1
data$ever_married[data$ever_married == "No"] <- 0 # no --> 0
```

Check dataset after all transformations

```
str(data)
```

```
## Classes 'data.table' and 'data.frame':   5109 obs. of  11 variables:
##  $ gender           : chr  "1" "0" "1" "0" ...
##  $ age              : num  67 61 80 49 79 81 74 69 59 78 ...
##  $ hypertension     : int  0 0 0 0 1 0 1 0 0 0 ...
##  $ heart_disease    : int  1 0 1 0 0 0 1 0 0 0 ...
##  $ ever_married     : chr  "1" "1" "1" "1" ...
##  $ work_type        : chr  "Private" "Self-employed" "Private" "Private" ...
##  $ Residence_type   : chr  "1" "0" "0" "1" ...
##  $ avg_glucose_level: num  229 202 106 171 174 ...
##  $ bmi              : num  36.6 29.1 32.5 34.4 24 ...
##  $ smoking_status   : chr  "formerly smoked" "never smoked" "never smoked" "smokes" ...
##  $ stroke           : int  1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
head(data)
```

```
##    gender age hypertension heart_disease ever_married     work_type
## 1:      1  67            0             1            1       Private
## 2:      0  61            0             0            1 Self-employed
## 3:      1  80            0             1            1       Private
## 4:      0  49            0             0            1       Private
## 5:      0  79            1             0            1 Self-employed
## 6:      1  81            0             0            1       Private
##    Residence_type avg_glucose_level      bmi  smoking_status stroke
## 1:              1            228.69 36.60000 formerly smoked      1
## 2:              0            202.21 29.06576    never smoked      1
## 3:              0            105.92 32.50000    never smoked      1
## 4:              1            171.23 34.40000          smokes      1
## 5:              0            174.12 24.00000    never smoked      1
## 6:              1            186.21 29.00000 formerly smoked      1
```

Just transformed variables are not yet numeric. For further analysis, the variables just transformed are converted into numerical values

```
suppressWarnings(data$gender <- as.numeric(as.character(data$gender)))
suppressWarnings(data$Residence_type <- as.numeric(as.character(data$Residence_type)))
suppressWarnings(data$ever_married <- as.numeric(as.character(data$ever_married)))
```

Further visual analysis of the data in connection with stoke, which has not yet been visually examined for correlation

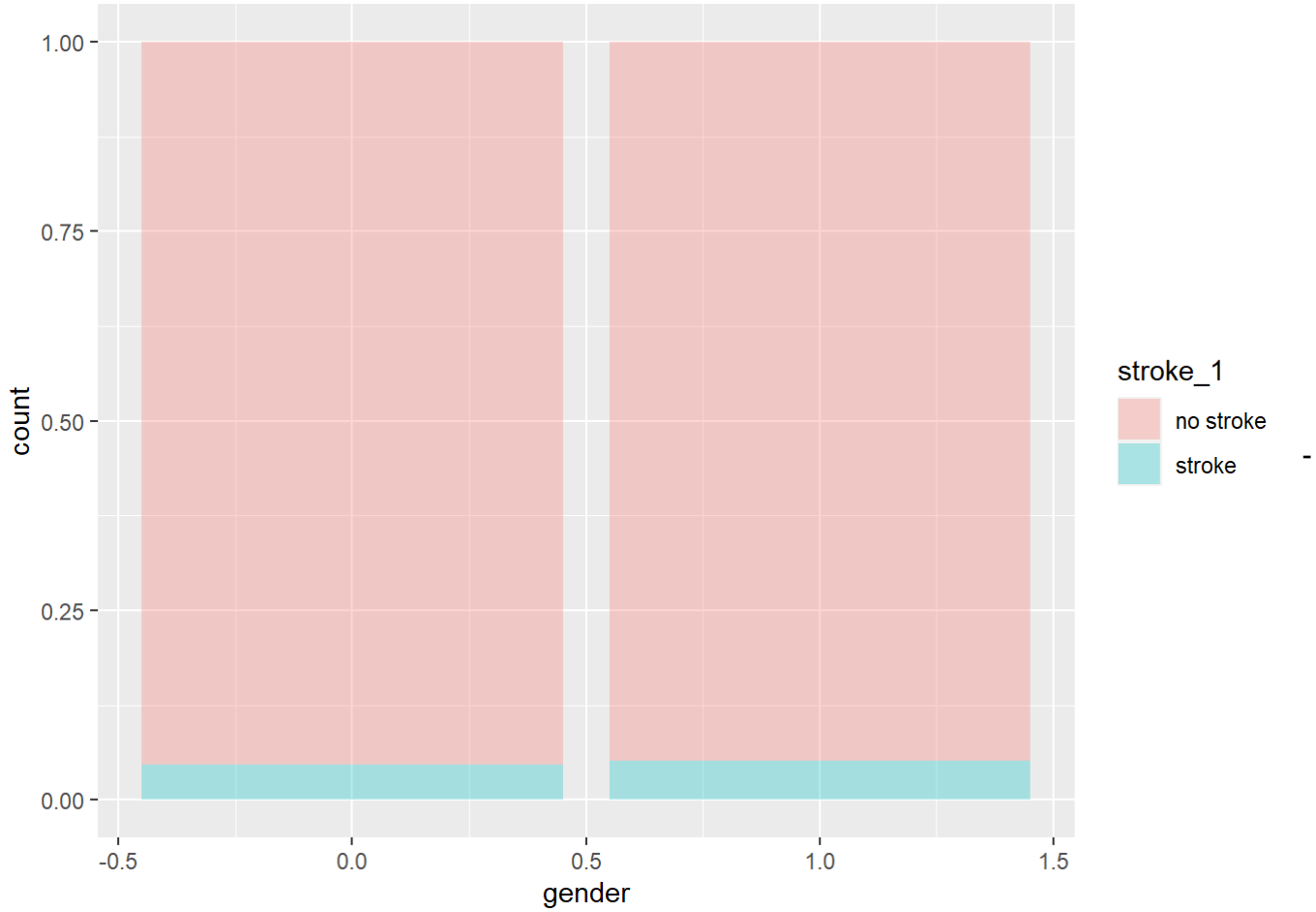
The goal is to find out which characteristic of the variables could have an influence on stroke

Create new stroke variable and transform variable as factor

```
data$stroke_1 = ifelse(data$stroke == 1, 'stroke', 'no stroke') # create new stroke variable
data$stroke_1 = factor(data$stroke_1) # transform new variable as factor
```

Combination of gender and stroke

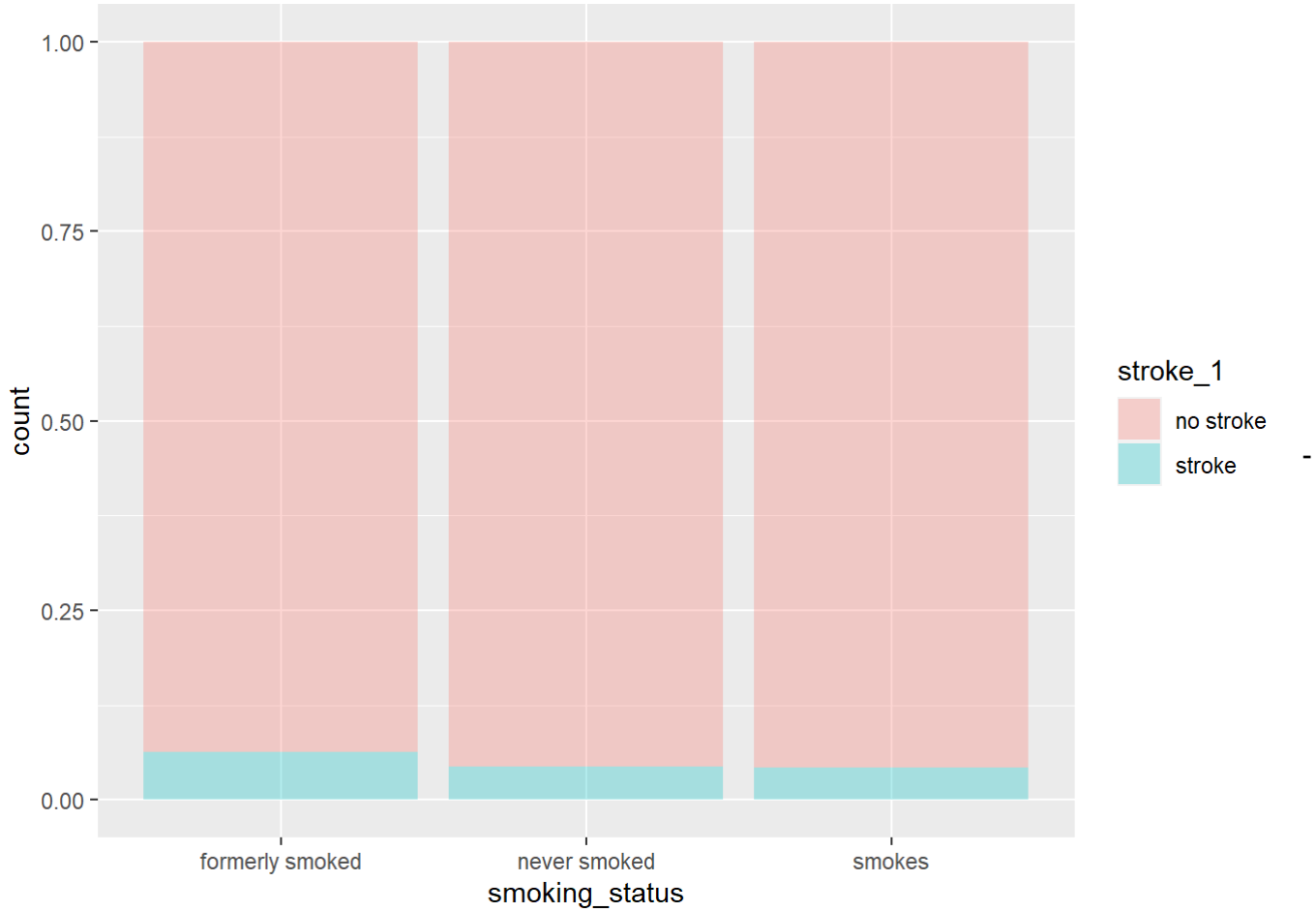
```
ggplot(data, aes(x = gender, fill = stroke_1))+ geom_bar(position = "fill" , alpha = 0.3)
```



> no clear visual impact

Combination of smoke status and stroke

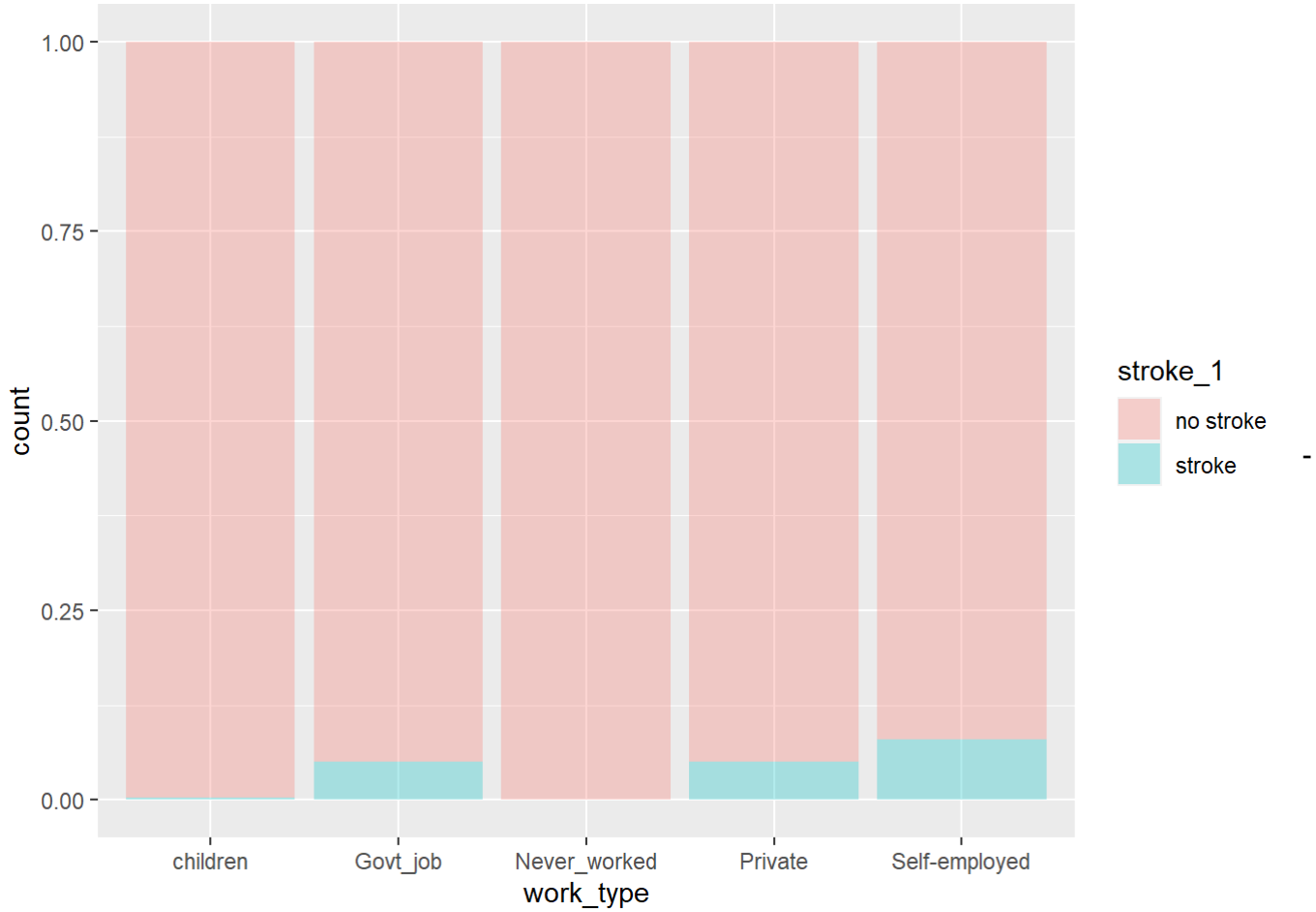
```
ggplot(data, aes(x = smoking_status, fill = stroke_1))+ geom_bar(position = "fill" , alpha = 0.3)
```



> no clear visual impact

Combination of work type and stroke

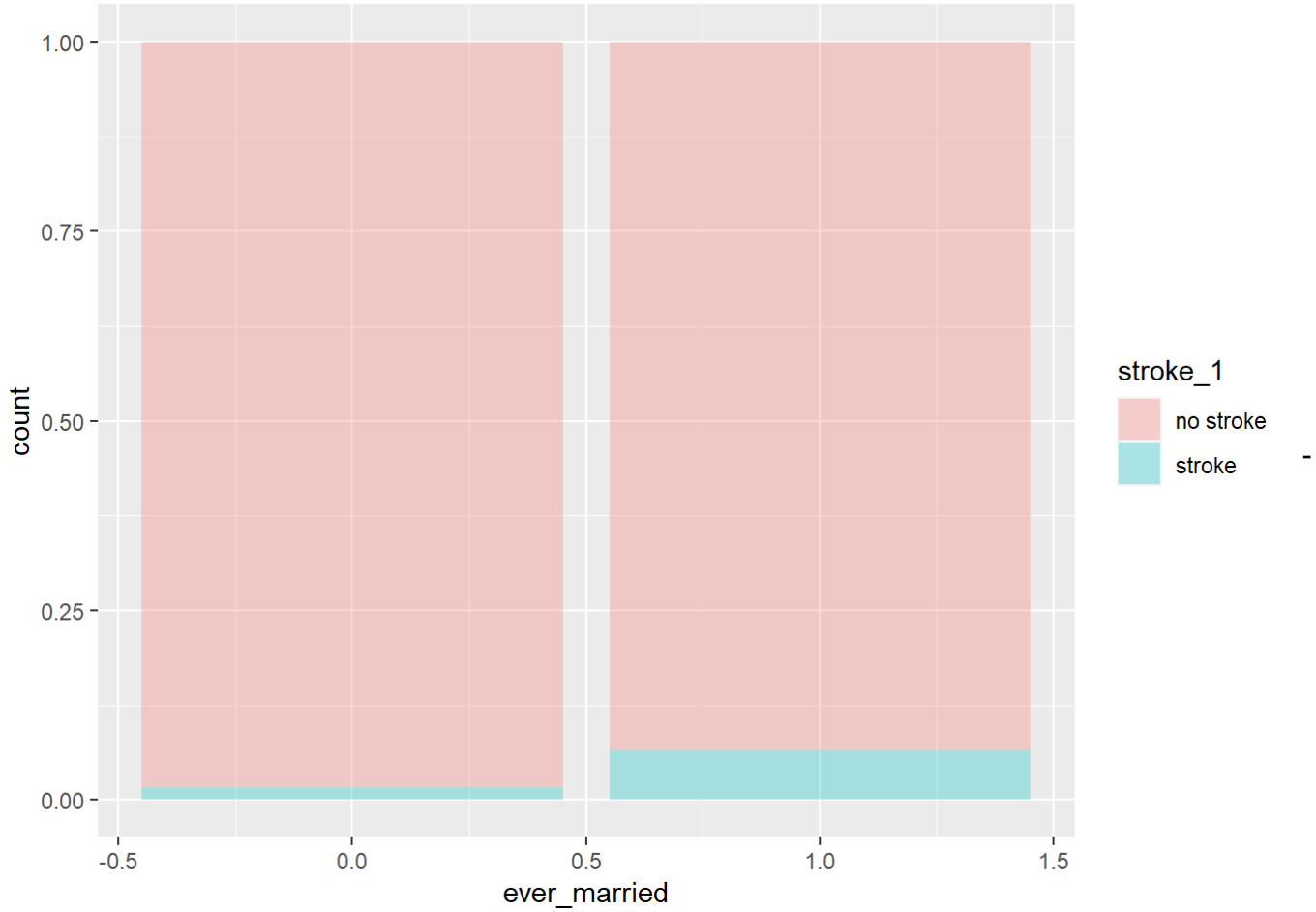
```
ggplot(data, aes(x = work_type, fill = stroke_1))+ geom_bar(position = "fill" , alpha = 0.3)
```



> work types “self-employed”, “Govt_job” and “Private” could have impact

Combination of bmi and stroke

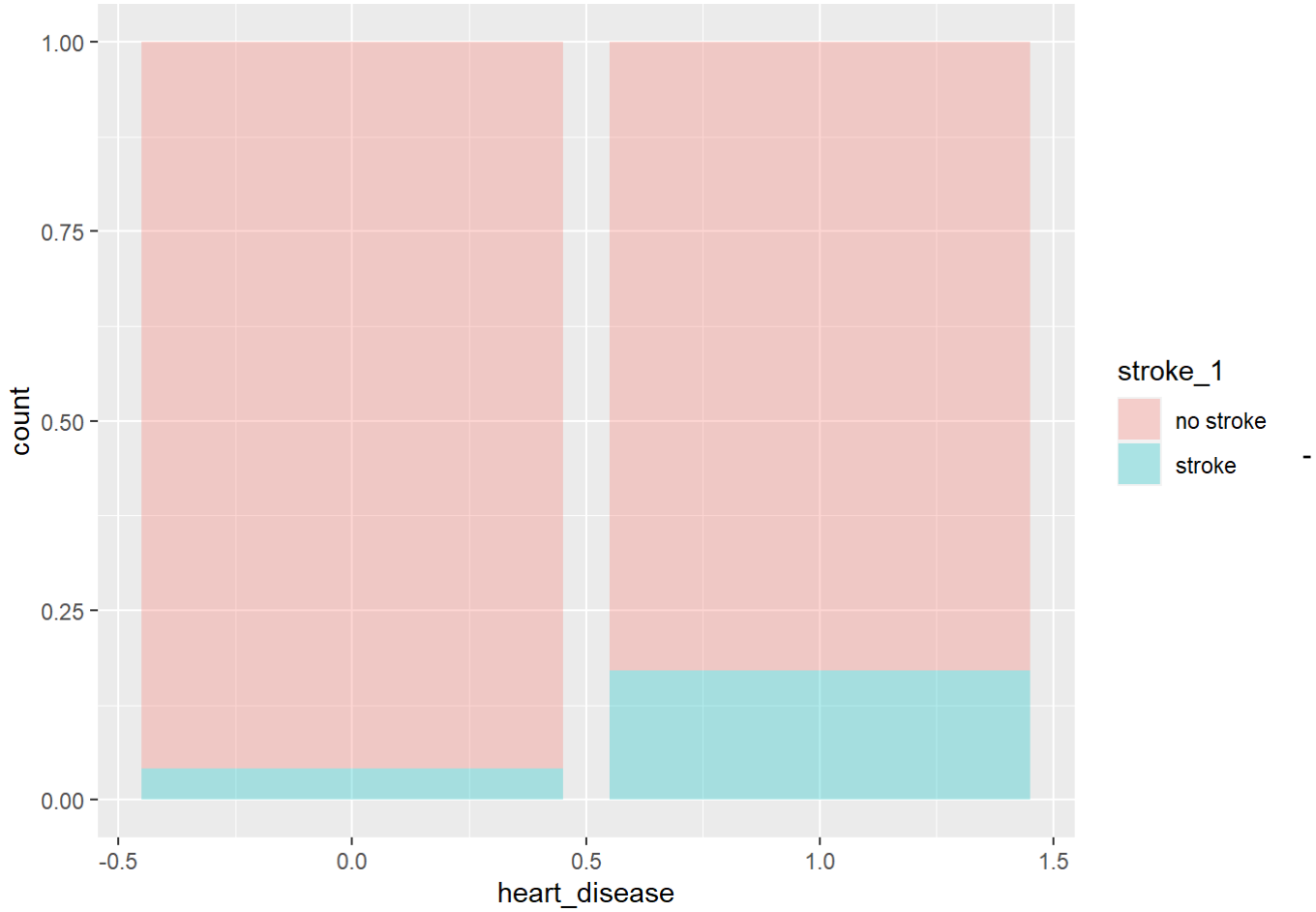
```
ggplot(data, aes(x = ever_married, fill = stroke_1))+ geom_bar(position = "fill" , alpha = 0.
3)
```



> marriage could have impact

Combination of bmi and stroke

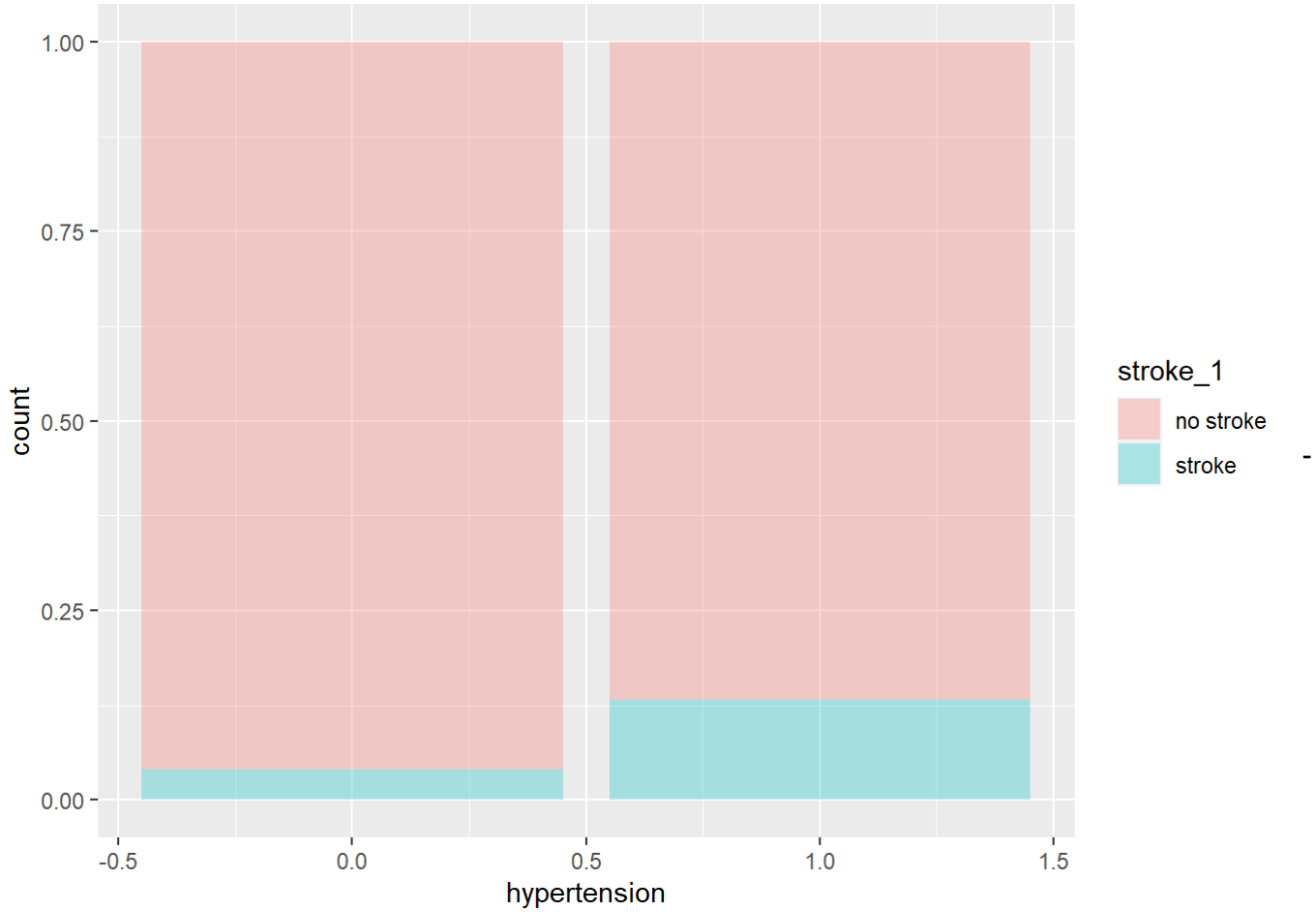
```
ggplot(data, aes(x = heart_disease, fill = stroke_1))+ geom_bar(position = "fill" , alpha = 0.3)
```



> heart disease could have impact

Combination of bmi and stroke

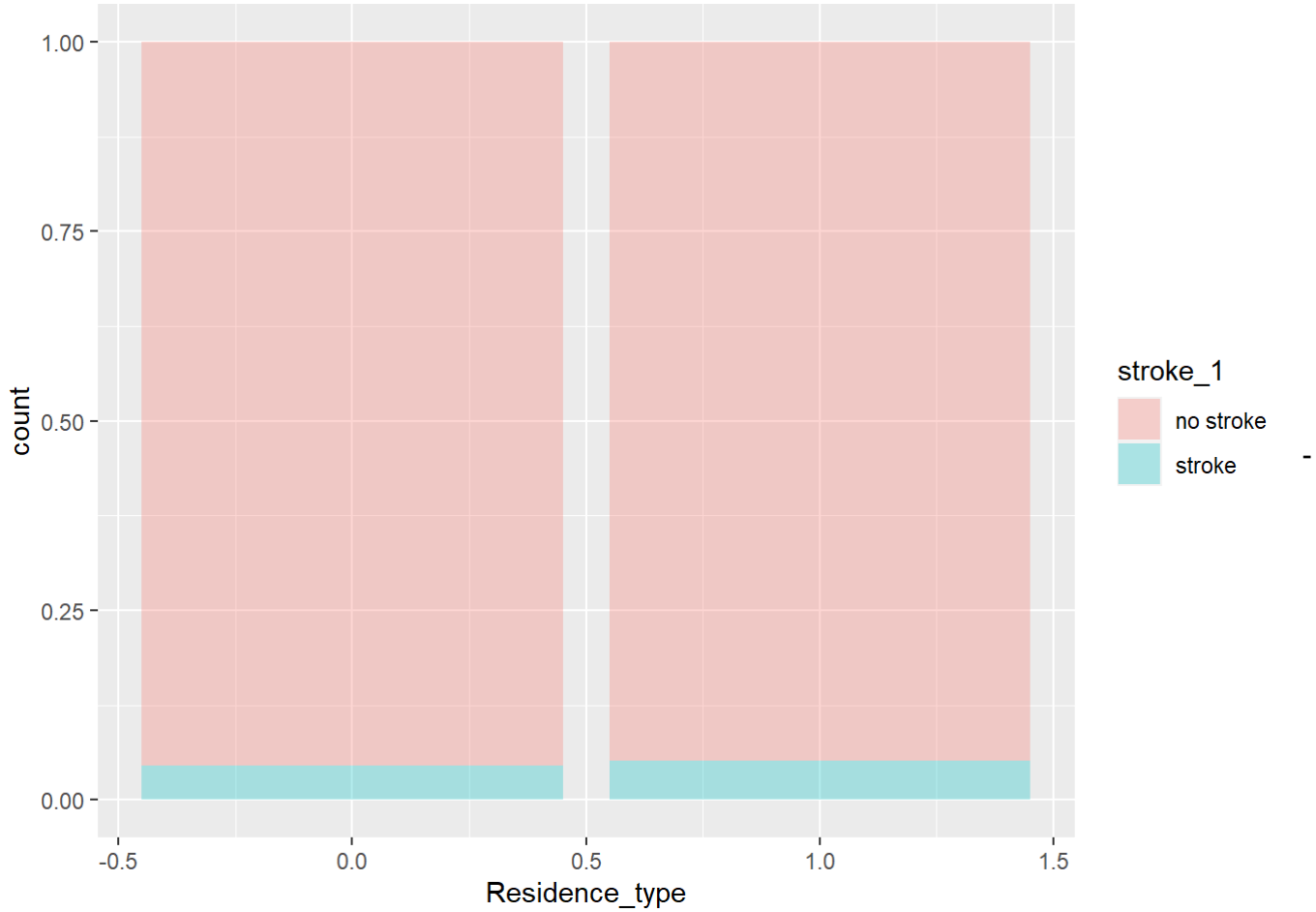
```
ggplot(data, aes(x = hypertension, fill = stroke_1))+ geom_bar(position = "fill" , alpha = 0.
3)
```



> hypertension could have impact

Combination of bmi and stroke

```
ggplot(data, aes(x = Residence_type, fill = stroke_1))+ geom_bar(position = "fill" , alpha = 0.3)
```



> no clear visual impact

Remove the just added support column stroke_1

```
data = subset(data, select = -c(stroke_1))
```

The final step of data exploration is to examine the correlation of all data For this analysis, all data must be available in numerical form, we see that there is data, that is not yet numerical

```
str(data)
```

```
## Classes 'data.table' and 'data.frame':   5109 obs. of  11 variables:
##  $ gender           : num  1 0 1 0 0 1 1 0 0 0 ...
##  $ age              : num  67 61 80 49 79 81 74 69 59 78 ...
##  $ hypertension     : int  0 0 0 0 1 0 1 0 0 0 ...
##  $ heart_disease    : int  1 0 1 0 0 0 1 0 0 0 ...
##  $ ever_married     : num  1 1 1 1 1 1 1 0 1 1 ...
##  $ work_type        : chr  "Private" "Self-employed" "Private" "Private" ...
##  $ Residence_type   : num  1 0 0 1 0 1 0 1 0 1 ...
##  $ avg_glucose_level: num  229 202 106 171 174 ...
##  $ bmi              : num  36.6 29.1 32.5 34.4 24 ...
##  $ smoking_status   : chr  "formerly smoked" "never smoked" "never smoked" "smokes" ...
##  $ stroke           : int  1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
data$hypertension = as.numeric(as.character(data$hypertension)) # transform to numerical data
data$heart_disease = as.numeric(as.character(data$heart_disease)) # transform to numerical da
ta
```

Replace text values within the variables and transform the values to numerical data

```
data$work_type = str_replace_all(data$work_type, c("Never_worked"="0","children"="1", "Private"="2", "Self-employed"="3", "Govt_job"="4")) # replace text with numbers
data$work_type = as.numeric(data$work_type) # transform to numerical data

data$smoking_status = str_replace_all(data$smoking_status, c("never smoked"="0","formerly smoked"="1", "smokes"="2")) # replace text with numbers
data$smoking_status = as.numeric(data$smoking_status) # transform to numerical data

data$stroke = as.numeric(as.character(data$stroke))
```

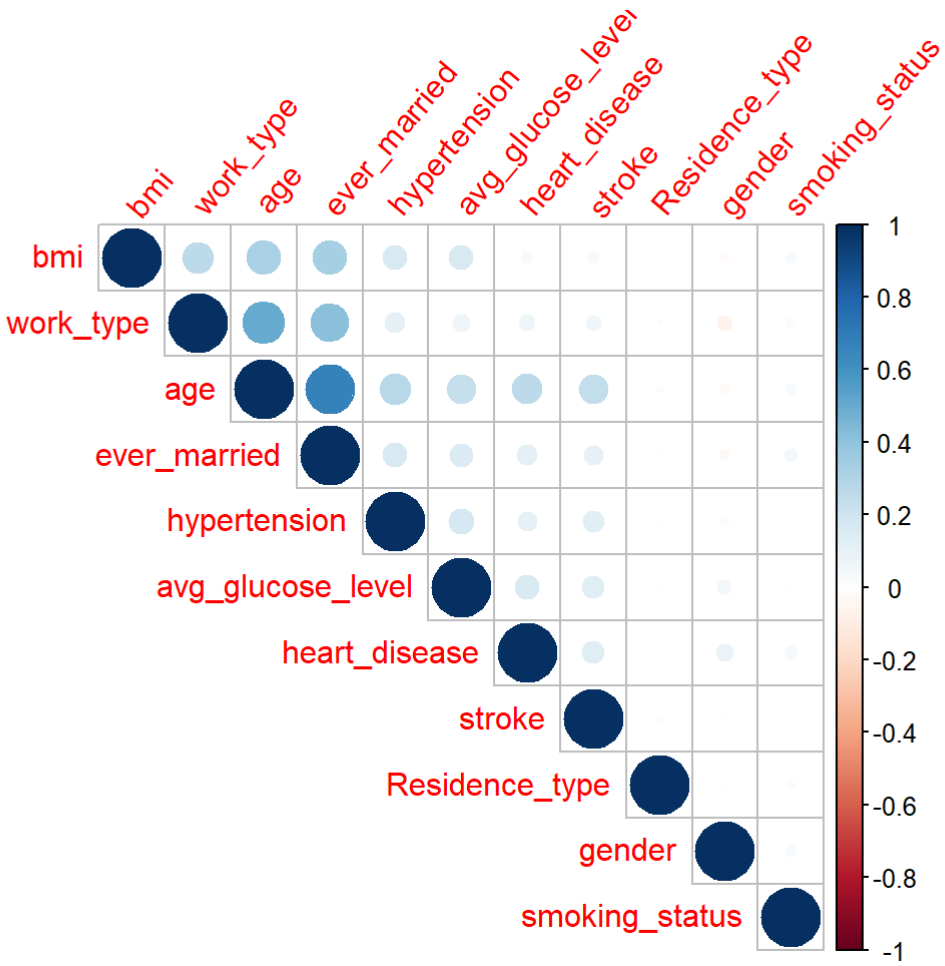
Recheck if all date is numerical and ready for last correlation analysis

```
str(data)
```

```
## Classes 'data.table' and 'data.frame':   5109 obs. of  11 variables:
##  $ gender           : num  1 0 1 0 0 1 1 0 0 0 ...
##  $ age              : num  67 61 80 49 79 81 74 69 59 78 ...
##  $ hypertension     : num  0 0 0 0 1 0 1 0 0 0 ...
##  $ heart_disease    : num  1 0 1 0 0 0 1 0 0 0 ...
##  $ ever_married     : num  1 1 1 1 1 1 1 0 1 1 ...
##  $ work_type        : num  2 3 2 2 3 2 2 2 2 2 ...
##  $ Residence_type   : num  1 0 0 1 0 1 0 1 0 1 ...
##  $ avg_glucose_level: num  229 202 106 171 174 ...
##  $ bmi              : num  36.6 29.1 32.5 34.4 24 ...
##  $ smoking_status   : num  1 0 0 2 0 1 0 0 2 1 ...
##  $ stroke           : num  1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Examine the data set for correlation

```
correlation <- cor(data)
corrplot(correlation, type = "upper", order = "hclust", tl.srt = 50)
```



One can see that it works. for example, there is a high correlation between married status and age As in the previous single variable analysis, there is no particularly strong correlation between stoke and any particular variable. However, it can be seen that age, hypertension, glucose lever heart disease, and marriage status may show a correlation.

Modeling

Logistic Regression

Logistic regression is used to make predictions about categorical variables, whereas linear regression is used to make predictions about a continuous variable. The model should predict whether a stroke occurs or not. Since the result of the prediction can only take two forms, stroke or no stroke (categorical variable), a logistic regression is used.

Split data in training and test data with 80% training data

```
set.seed(5)
test_index <- createDataPartition(data$stroke, times = 1, p = 0.8, list = FALSE)
test <- data[-test_index, ]
train <- data[test_index, ]

dim(train)
```

```
## [1] 4088   11
```

```
dim(test)
```

```
## [1] 1021   11
```

Check the probability of stoke in the two datasets to ensure that the split is usable

```
prop.table(table(train$stroke))
```

```
##
##          0          1
## 0.95401174 0.04598826
```

```
prop.table(table(test$stroke))
```

```
##
##          0          1
## 0.94025465 0.05974535
```

The probabilities are close together, so the datasets are appropriate

Create Generalized linear model with family = binomial

```
glm_regression <- glm(stroke~., data=train, family=binomial)
```

Check the model to see which influencing factors the model sees

```
summary(glm_regression)
```

```
##
## Call:
## glm(formula = stroke ~ ., family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0382  -0.3127  -0.1677  -0.0826   3.7357
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -7.252431   0.638036 -11.367  < 2e-16 ***
## gender             0.038225   0.161461   0.237  0.81286
## age                0.068748   0.006070  11.326  < 2e-16 ***
## hypertension       0.456291   0.187016   2.440  0.01469 *
## heart_disease      0.202055   0.221431   0.912  0.36151
## ever_married       0.034213   0.271014   0.126  0.89954
## work_type         -0.162525   0.112924  -1.439  0.15008
## Residence_type     0.119955   0.158359   0.757  0.44876
## avg_glucose_level  0.004218   0.001364   3.091  0.00199 **
## bmi               -0.002392   0.012936  -0.185  0.85327
## smoking_status     0.123873   0.101740   1.218  0.22340
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1525.1  on 4087  degrees of freedom
## Residual deviance: 1223.2  on 4077  degrees of freedom
## AIC: 1245.2
##
## Number of Fisher Scoring iterations: 7
```

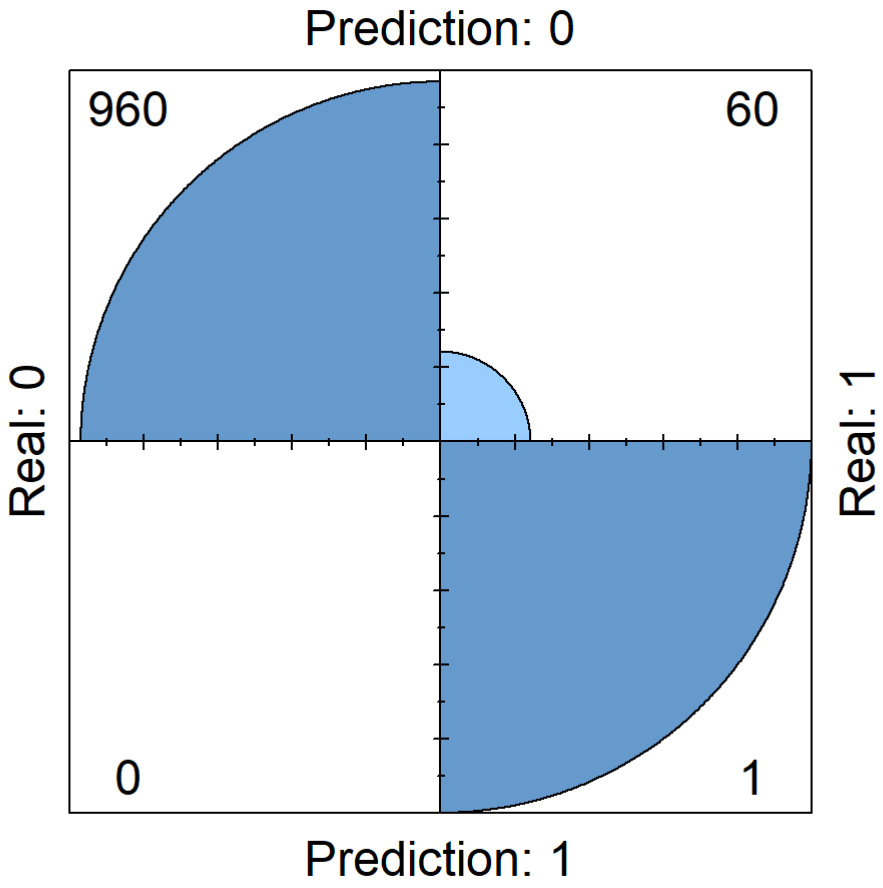
The influencing factors found by the model are consistent with those from the previous corellation analysis

Test the model using the test set

```
prediction <- predict(glm_regression, test, type="response")
```

Check prediction results

```
pred_test <- ifelse(prediction >0.5,1,0) # if prediction over 0.5 than stroke prediction
fourfoldplot(table(Prediction = pred_test, Real = test$stroke), conf.level = 0, margin = 1) #
check confusion matrix -- > High number of false negative, few true positives
```




```
print(1-mean(pred_test != test$stroke)) # High Accuracy
```

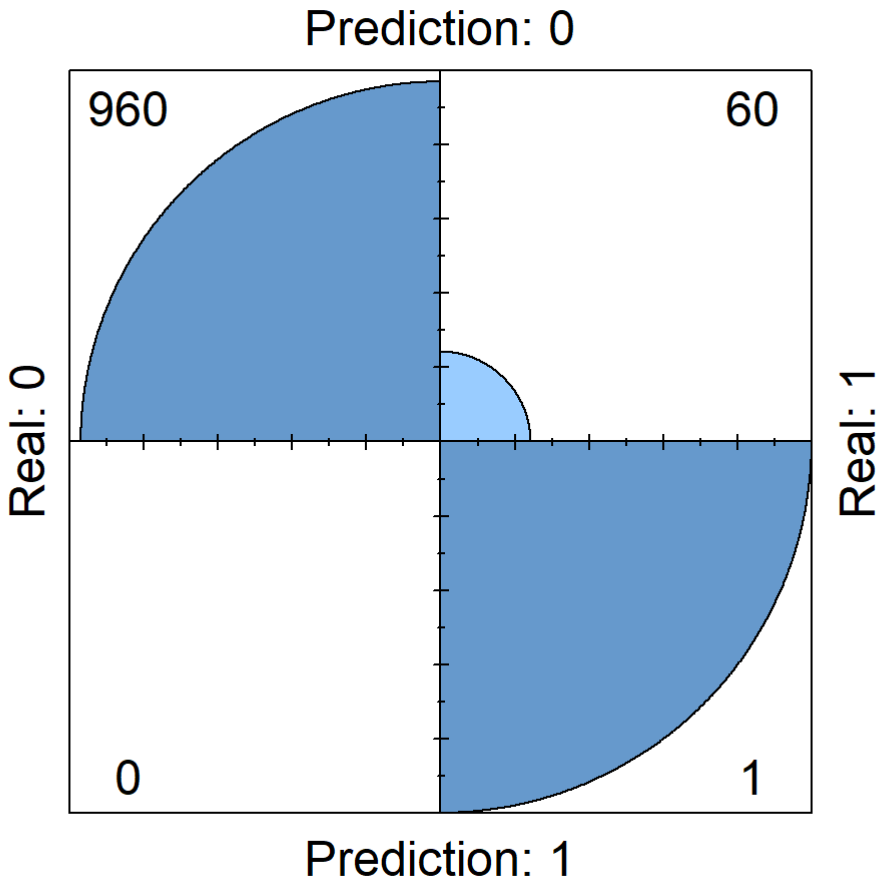
```
## [1] 0.9412341
```

The results show a high number of false negative, few true positives and high Accuracy

The threshold from which a stoke is predicted is apparently too high, therefore test with a different threshold

Prediction threshold 0.4

```
pred_test <- ifelse(prediction >0.4,1,0)
fourfoldplot(table(Prediction = pred_test, Real = test$stroke), conf.level = 0, margin = 1) #
check confusion matrix -- > no change to previous setting
```



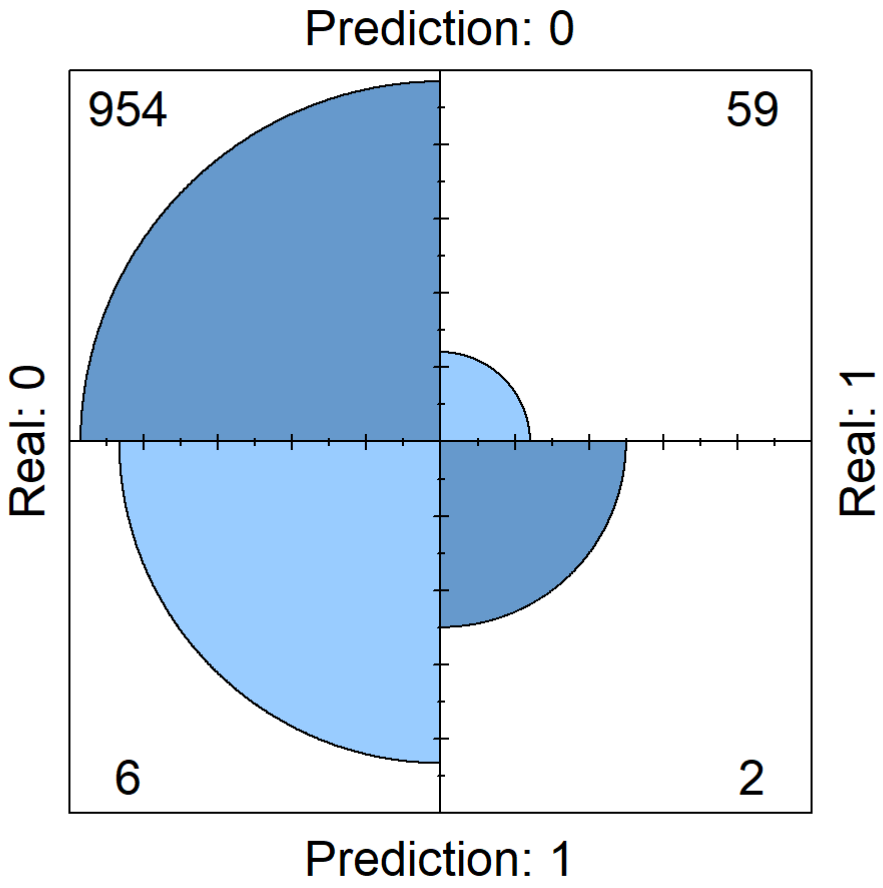

```
print(1-mean(pred_test != test$stroke)) # same high Accuracy as before
```

```
## [1] 0.9412341
```

The results show a high number of false negative, few true positives and high Accuracy –> same as with previous settings

Prediction threshold 0.3

```
pred_test <- ifelse(prediction >0.3,1,0)
fourfoldplot(table(Prediction = pred_test, Real = test$stroke), conf.level = 0, margin = 1) # check confusion matrix -- > more true positives, but also more false positives
```



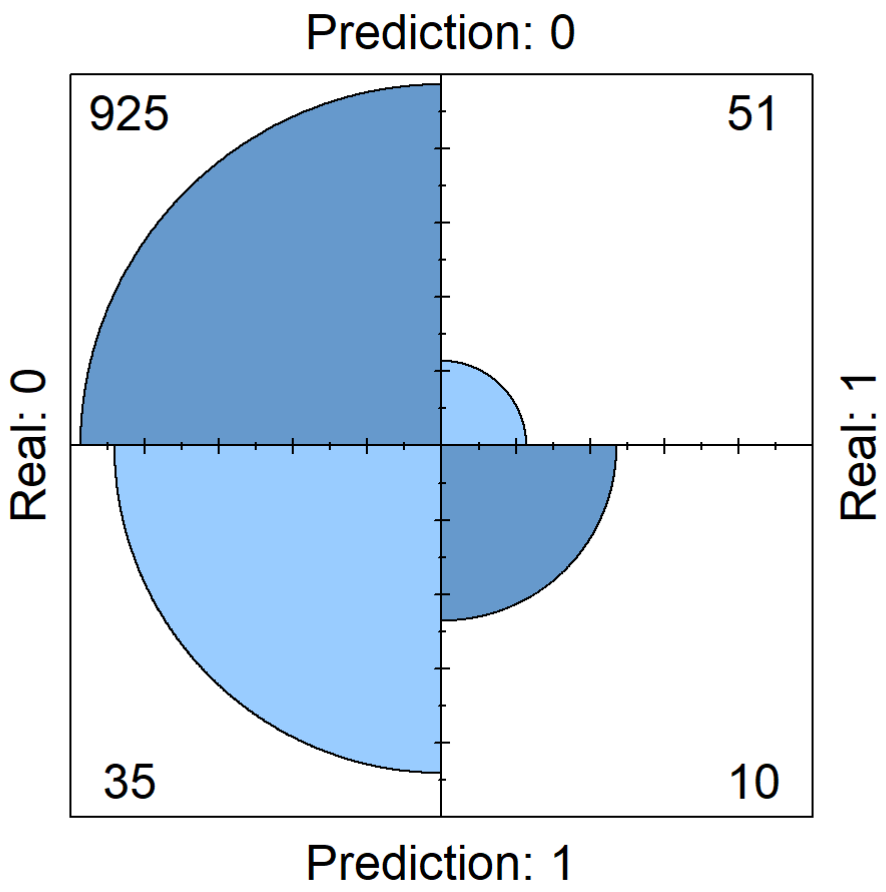

```
print(1-mean(pred_test != test$stroke)) # Accuracy just litte changed
```

```
## [1] 0.9363369
```

The results show more true positives, approixmatly the same number of false negatives but also more false positives with just marginal changes in accuracy

Prediction threshold 0.2

```
pred_test <- ifelse(prediction >0.2,1,0)
fourfoldplot(table(Prediction = pred_test, Real = test$stroke), conf.level = 0, margin = 1) #
check confusion matrix -- > more true positives, fewer false negatives
```



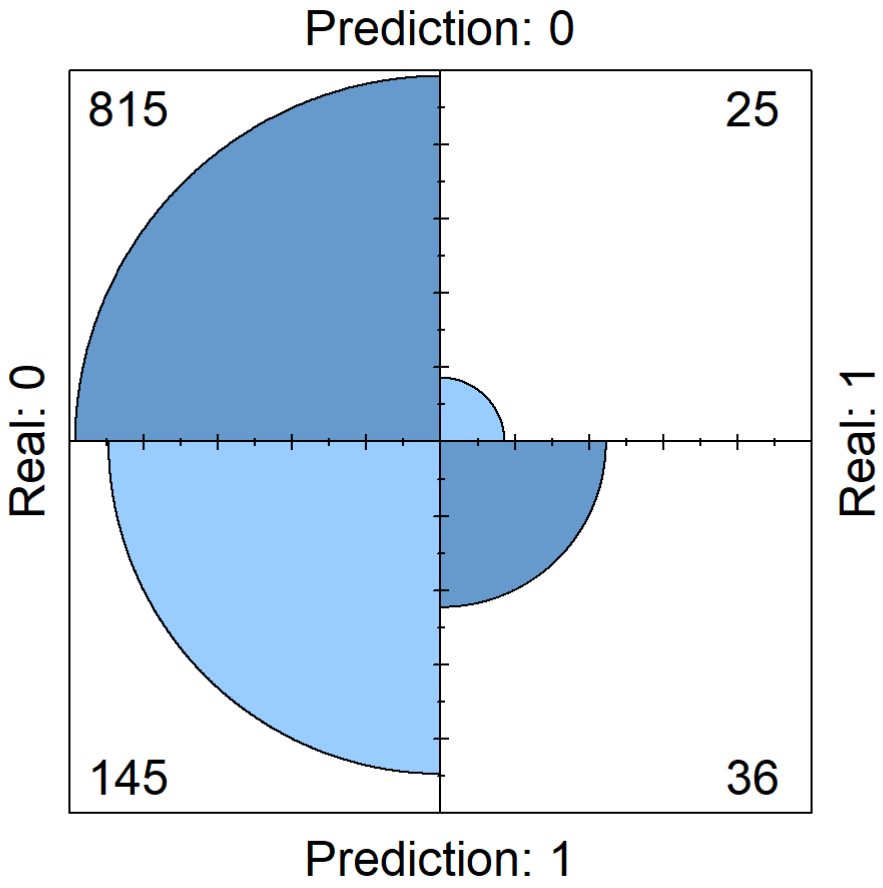

```
print(1-mean(pred_test != test$stroke)) # Accuracy lower than before, but still good
```

```
## [1] 0.9157689
```

The results show again more true positives, fewer false negatives but more false positives and a lower accuracy than before

Prediction threshold 0.1

```
pred_test <- ifelse(prediction >0.1,1,0)
fourfoldplot(table(Prediction = pred_test, Real = test$stroke), conf.level = 0, margin = 1) # highest number of true positves and lowest number of false negatives.
```




```
print(1-mean(pred_test != test$stroke)) # Accuracy lower than before, but still ok
```

```
## [1] 0.8334966
```

With this setting the numbers of false positives and true positives are the highest, but the number of false negatives is the lowest.

In this case, false positives are less bad than false negatives, so these settings will be used

For the fact that the number of true positives goes up, the number of false positives also goes up. In this case, however, it is better to predict false positives and have more true positives than to have fewer false positives and fewer true positives

At the expense of Accuracy the number of true positives increases With these results, the model could be used as a kind of warning system in this case

Test Model with less variable

Based on the previous model summary and the performed data exploration one can see that age, hypertension, heard disease and glucose levels could have biggest impact out of all variables

Next step ist to try a models with only these variables an see if the pervious results could be increased

Create new dataset for this test

```
data1 <- data
data1 = subset(data1, select = -c(gender, ever_married, work_type, Residence_type, bmi, smoki
ng_status))
```

check newly created dataset

```
str(data1)
```

```
## Classes 'data.table' and 'data.frame':   5109 obs. of  5 variables:
##  $ age              : num  67 61 80 49 79 81 74 69 59 78 ...
##  $ hypertension     : num  0 0 0 0 1 0 1 0 0 0 ...
##  $ heart_disease    : num  1 0 1 0 0 0 1 0 0 0 ...
##  $ avg_glucose_level: num  229 202 106 171 174 ...
##  $ stroke           : num  1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Split data in training and test data with 80% training data

```
set.seed(5)
test_index <- createDataPartition(data1$stroke, times = 1, p = 0.8, list = FALSE)
test1 <- data1[-test_index, ]
train1 <- data1[test_index, ]
```

Check the probability of stoke in the two datasets to ensure that the split is usable

```
prop.table(table(train1$stroke))
```

```
##
##          0          1
## 0.95401174 0.04598826
```

```
prop.table(table(test1$stroke))
```

```
##
##          0          1
## 0.94025465 0.05974535
```

Create Generalized linear model with family = binomial

```
glm_regression1 <- glm(stroke~., data=train1, family=binomial)
```

Check the model to see which influencing factors the model sees

```
summary(glm_regression1)
```

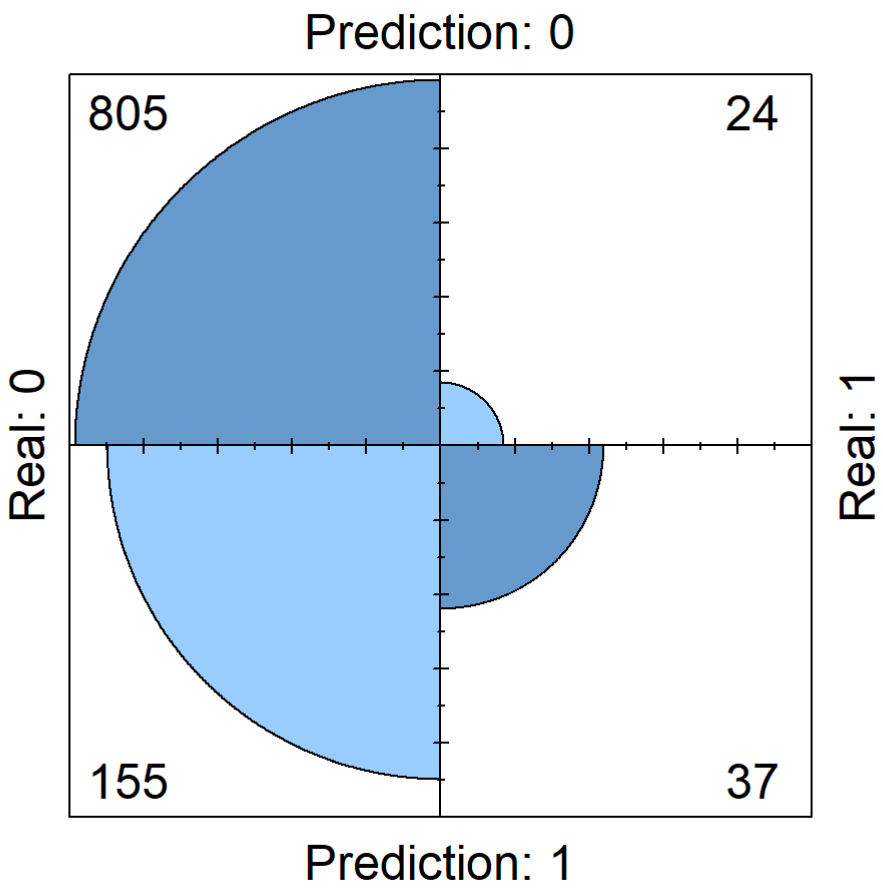
```
##
## Call:
## glm(formula = stroke ~ ., family = binomial, data = train1)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0211  -0.3142  -0.1716  -0.0842   3.7581
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -7.448094   0.400933 -18.577  < 2e-16 ***
## age                0.067326   0.005809  11.591  < 2e-16 ***
## hypertension       0.431554   0.185715   2.324  0.02014 *
## heart_disease      0.247590   0.216680   1.143  0.25318
## avg_glucose_level  0.004243   0.001326   3.200  0.00138 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1525.1  on 4087  degrees of freedom
## Residual deviance: 1227.7  on 4083  degrees of freedom
## AIC: 1237.7
##
## Number of Fisher Scoring iterations: 7
```

Test the model using the test set

```
prediction1 <- predict(glm_regression1, test1, type="response")
```

Check prediction results

```
pred_test1 <- ifelse(prediction1 >0.1,1,0) # use best setting from first model
fourfoldplot(table(Prediction = pred_test1, Real = test1$stroke), conf.level = 0, margin = 1)
```




```
print(1-mean(pred_test1 != test1$stroke))
```

```
## [1] 0.8246817
```

Both the confusion matix and the accuracy are almost equal to the previous test. The model could therefore not be improved in this way. But the result show that the variables that were dropped in this case have no real influence on the model.

Stepwise AIC

Next, the stepAIC is tested to automaticly perform the previous manual step and try the model with differnet cominations of the variables. The goal is to minimize the stepAIC value to come up with a reduced set of variablethe for the final model This approach does not automatically mean that the performance of the model is improved, but is used to simplify the model without significantly affecting its performance. The Dataset from fist model is used, so that the model can chose from all variables

```
glm_regression_steps = glm(stroke~., data=train, family = "binomial") %>% stepAIC(trace = TRU
E)
```

```
## Start:  AIC=1245.21
## stroke ~ gender + age + hypertension + heart_disease + ever_married +
##     work_type + Residence_type + avg_glucose_level + bmi + smoking_status
##
##                     Df Deviance    AIC
## - ever_married       1   1223.2 1243.2
## - bmi                1   1223.2 1243.2
## - gender             1   1223.3 1243.3
## - Residence_type     1   1223.8 1243.8
## - heart_disease      1   1224.0 1244.0
## - smoking_status     1   1224.7 1244.7
## <none>                   1223.2 1245.2
## - work_type          1   1225.3 1245.3
## - hypertension       1   1228.9 1248.9
## - avg_glucose_level  1   1232.5 1252.5
## - age                1   1376.7 1396.7
##
## Step:  AIC=1243.22
## stroke ~ gender + age + hypertension + heart_disease + work_type +
##     Residence_type + avg_glucose_level + bmi + smoking_status
##
##                     Df Deviance    AIC
## - bmi                1   1223.3 1241.3
## - gender             1   1223.3 1241.3
## - Residence_type     1   1223.8 1241.8
## - heart_disease      1   1224.0 1242.0
## - smoking_status     1   1224.7 1242.7
## <none>                   1223.2 1243.2
## - work_type          1   1225.3 1243.3
## - hypertension       1   1228.9 1246.9
## - avg_glucose_level  1   1232.5 1250.5
## - age                1   1407.8 1425.8
##
## Step:  AIC=1241.26
## stroke ~ gender + age + hypertension + heart_disease + work_type +
##     Residence_type + avg_glucose_level + smoking_status
##
##                     Df Deviance    AIC
## - gender             1   1223.3 1239.3
## - Residence_type     1   1223.8 1239.8
## - heart_disease      1   1224.1 1240.1
## - smoking_status     1   1224.7 1240.7
## <none>                   1223.3 1241.3
## - work_type          1   1225.4 1241.4
## - hypertension       1   1228.9 1244.9
## - avg_glucose_level  1   1232.7 1248.7
## - age                1   1407.9 1423.9
##
## Step:  AIC=1239.32
## stroke ~ age + hypertension + heart_disease + work_type + Residence_type +
##     avg_glucose_level + smoking_status
##
##                     Df Deviance    AIC
## - Residence_type     1   1223.9 1237.9
## - heart_disease      1   1224.2 1238.2
```

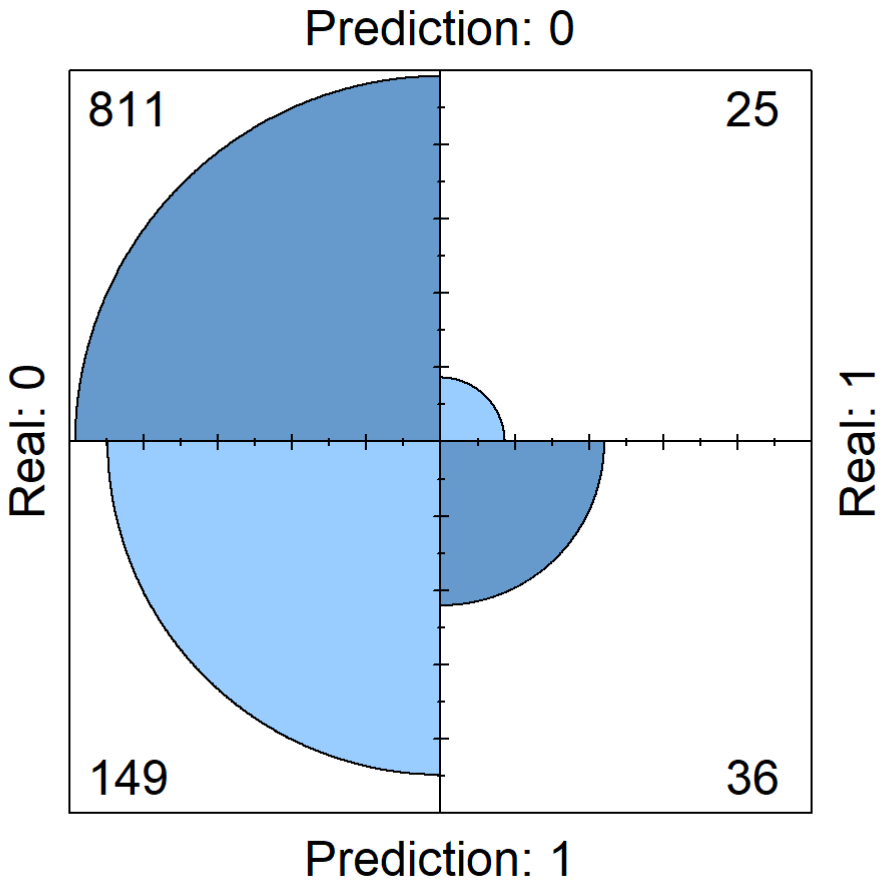
```
## - smoking_status     1   1224.9 1238.9
## <none>                   1223.3 1239.3
## - work_type          1   1225.5 1239.5
## - hypertension       1   1229.0 1243.0
## - avg_glucose_level  1   1233.0 1247.0
## - age                1   1407.9 1421.9
##
## Step:  AIC=1237.88
## stroke ~ age + hypertension + heart_disease + work_type + avg_glucose_level +
##     smoking_status
##
##                     Df Deviance    AIC
## - heart_disease      1   1224.7 1236.7
## - smoking_status     1   1225.6 1237.6
## <none>                   1223.9 1237.9
## - work_type          1   1226.0 1238.0
## - hypertension       1   1229.4 1241.4
## - avg_glucose_level  1   1233.6 1245.6
## - age                1   1409.3 1421.3
##
## Step:  AIC=1236.72
## stroke ~ age + hypertension + work_type + avg_glucose_level +
##     smoking_status
##
##                     Df Deviance    AIC
## <none>                   1224.7 1236.7
## - smoking_status     1   1226.8 1236.8
## - work_type          1   1226.9 1236.9
## - hypertension       1   1230.3 1240.3
## - avg_glucose_level  1   1235.0 1245.0
## - age                1   1428.9 1438.9
```

Test the Stepwise AIC model using the test set

```
prediction2 <- predict(glm_regression_steps, test, type="response") # predict with steps mode
l
```

Check prediction results

```
pred_test2 <- ifelse(prediction2 >0.1,1,0) # use the best setting from first model
fourfoldplot(table(Prediction = pred_test2, Real = test$stroke), conf.level = 0, margin = 1)
# Not much difference from the previous two attempts. The number of false positives drops onl
y minimally
```




```
print(1-mean(pred_test2 != test$stroke))
```

```
## [1] 0.8295788
```

The Stepwiese AIC model with an automatically reduced number of variables has not brought any major changes either. So all attempts to change the combination of variables did not bring much. The results all remain very similar

Oversampling

Since the number of stokes is very small compared to the number of non-strokes, another possibility is oversample the data to artificially adjust the number of strokes and non-strokes. The dataset that was used for the first model is now used again and oversampling is applied to it

Artificially increase the number of strokes

```
data2 <- ovun.sample(stroke~.,data = data, method = 'over',p = 0.3)$data # the number of stro
kes will artificially be increased
```

Before oversampling

```
table(data$stroke)
```

```
##
##    0    1
## 4860  249
```

After oversampling

```
table(data2$stroke)
```

```
##
##    0    1
## 4860 2104
```

–> number of strokes increased significant

Split data in training and test data with 80% training data

```
set.seed(5)
test_index <- createDataPartition(data2$stroke, times = 1, p = 0.8, list = FALSE)
test2 <- data2[-test_index, ]
train2 <- data2[test_index, ]
```

Check the probability of stoke in the two datasets to ensure that the split is usable

```
prop.table(table(train2$stroke))
```

```
##
##        0        1
## 0.701364 0.298636
```

```
prop.table(table(test2$stroke))
```

```
##
##        0        1
## 0.683908 0.316092
```

The probabilities are close together, so the datasets are appropriate. Also here one can see the result of the oversampling, because the probability for stoke in this data is much higher than in the previous.

Create Generalized linear model with family = binomial

```
glm_regression2 <- glm(stroke~., data=train2, family=binomial)
```

Check the model to see which influencing factors the model sees

```
summary(glm_regression2)
```

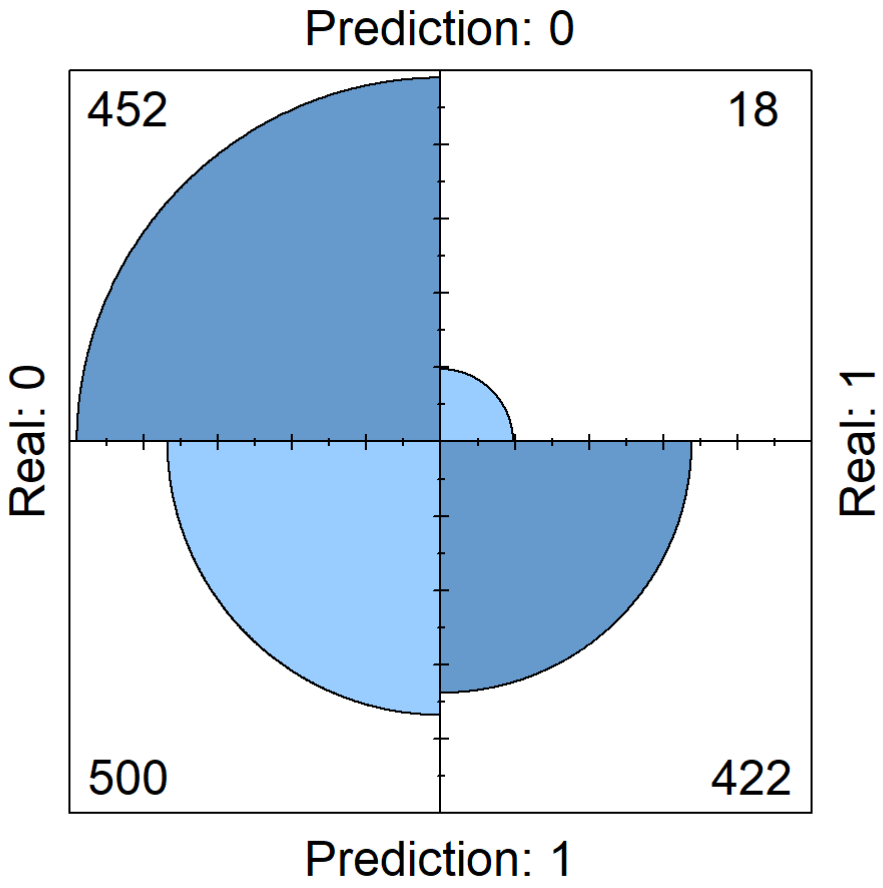
```
##
## Call:
## glm(formula = stroke ~ ., family = binomial, data = train2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2124  -0.6954  -0.3053   0.8045   3.0985
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -5.3336418  0.2612213 -20.418  < 2e-16 ***
## gender            -0.0267626  0.0728062  -0.368   0.7132
## age                0.0692164  0.0025771  26.858  < 2e-16 ***
## hypertension       0.5607995  0.0899209   6.237 4.47e-10 ***
## heart_disease      0.4720165  0.1113070   4.241 2.23e-05 ***
## ever_married      -0.2738383  0.1094510  -2.502   0.0124 *
## work_type         -0.0713341  0.0475181  -1.501   0.1333
## Residence_type     0.0049210  0.0710447   0.069   0.9448
## avg_glucose_level  0.0046437  0.0006564   7.075 1.50e-12 ***
## bmi                0.0065337  0.0054555   1.198   0.2311
## smoking_status     0.0810190  0.0460982   1.758   0.0788 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6794.5  on 5571  degrees of freedom
## Residual deviance: 4914.9  on 5561  degrees of freedom
## AIC: 4936.9
##
## Number of Fisher Scoring iterations: 5
```

Test the model using the test set

```
prediction3 <- predict(glm_regression2, test2, type="response")
```

Check prediction results

```
pred_test3 <- ifelse(prediction3 >0.1,1,0) # use the best setting from first model
fourfoldplot(table(Prediction = pred_test3, Real = test2$stroke), conf.level = 0, margin = 1)
```




```
print(1-mean(pred_test3 != test2$stroke))
```

```
## [1] 0.6278736
```

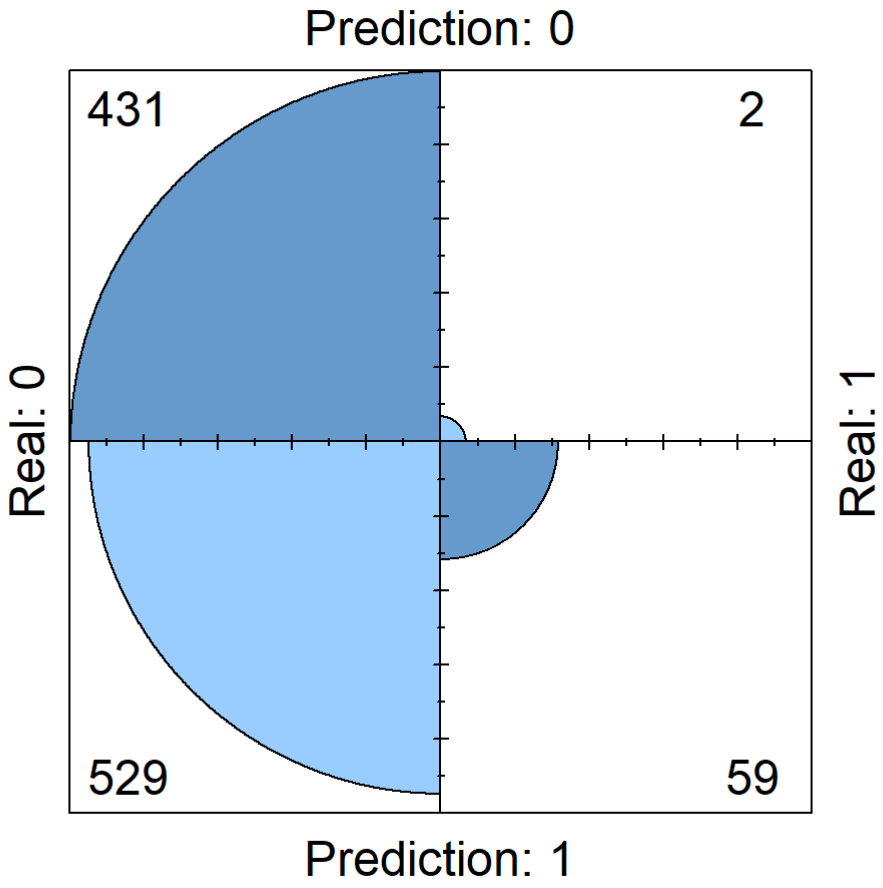
Even though it is not really comparable to the previous models without oversampling, the number of false negatives is lower compared to the models without oversampling, what is good from medical point of view The accuracy droped significantly

Testing the algorithm trained with oversampling with the test data without oversampling

```
prediction4 <- predict(glm_regression2, test, type="response")
```

Check prediction results

```
pred_test4 <- ifelse(prediction4 >0.1,1,0) # use the best setting from first model
fourfoldplot(table(Prediction = pred_test4, Real = test$stroke), conf.level = 0, margin = 1)
# Not much difference from the previous two attempts. The number of false positives drops onl
y minimally
```




```
print(1-mean(pred_test4 != test$stroke))
```

```
## [1] 0.4799216
```

In this case, the number of false negatives is very low and the number of true positives is the highest of all those tested. From a medical point of view, this model is therefore the best at first glance. Unfortunately, the number of false positives is also very high and the accuracy very low. So it looks like the first model is still the best so far

Random Forest

Next, the random forest model is tested, as this model can also be used to predict classifications and is used for input variables without much correlation. (as in as in the dataset at hand) This model is also advanced which is why better results are hoped for

The train and test dataset from the first model will be used, but the searched variable, stroke, is transformed into a factor

```
train$stroke <- as.character(train$stroke)
train$stroke <- as.factor(train$stroke)
test$stroke <- as.character(test$stroke)
test$stroke <- as.factor(test$stroke)
```

Create random forest model

```
ran_for = randomForest(stroke~., train, importance=TRUE)
summary(ran_for)
```

```
##                 Length Class  Mode
## call               4   -none- call
## type               1   -none- character
## predicted       4088   factor numeric
## err.rate        1500   -none- numeric
## confusion          6   -none- numeric
## votes           8176   matrix numeric
## oob.times       4088   -none- numeric
## classes            2   -none- character
## importance        40   -none- numeric
## importanceSD      30   -none- numeric
## localImportance    0   -none- NULL
## proximity          0   -none- NULL
## ntree              1   -none- numeric
## mtry               1   -none- numeric
## forest            14   -none- list
## y               4088   factor numeric
## test               0   -none- NULL
## inbag              0   -none- NULL
## terms              3   terms  call
```

```
ran_for
```

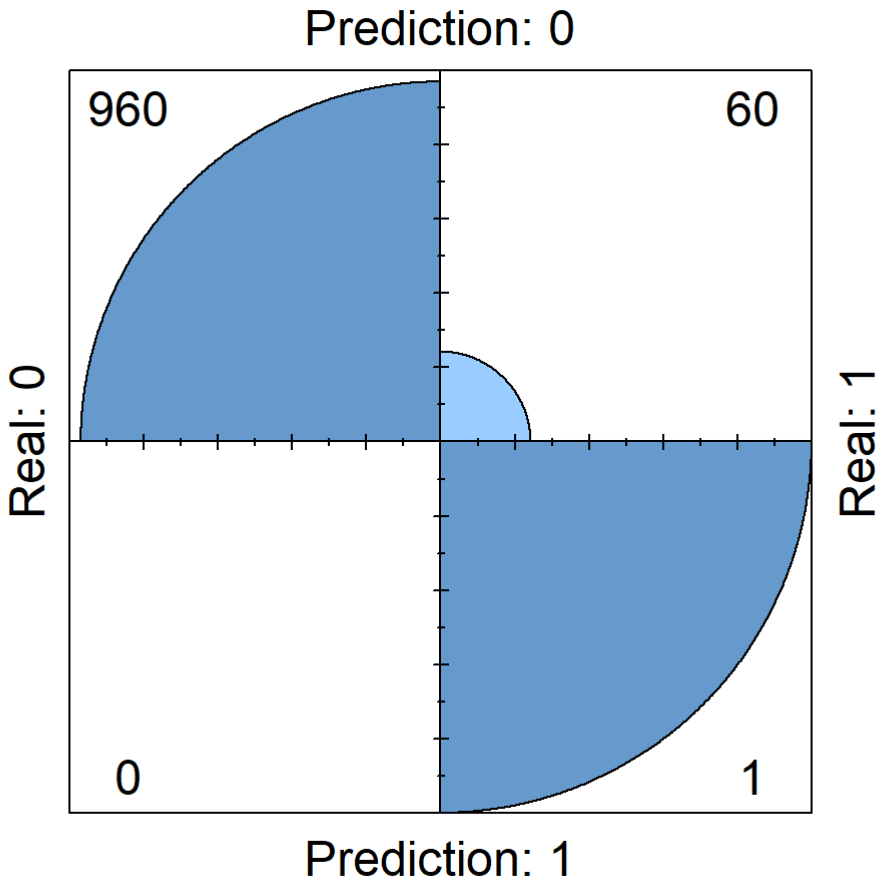
```
##
## Call:
##  randomForest(formula = stroke ~ ., data = train, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 4.72%
## Confusion matrix:
##      0 1 class.error
## 0 3894 6 0.001538462
## 1  187 1 0.994680851
```

Test model on test data

```
rf_predict = predict(ran_for, test)
```

Check prediction results

```
fourfoldplot(table(Prediction = rf_predict, Real = test$stroke), conf.level = 0, margin = 1)
```




```
print(1-mean(rf_predict != test$stroke))
```

```
## [1] 0.9412341
```

The model produces very similar results to the very first attempt at logistic regression. There are very few true positives, relatively many false negatives, but very high accuracy Since only very few ture positives are detected, the model is not particularly good from a medical point of view.

Test whether adjusting model variables can improve the model*

In this step the error rate should be minimized by finding optimal model variables

```
NTrees=1:10 # Number of trees (tried with higher nubers, but calculation will take super lon
g)
MTRY=1:10 # Number of variables (tried with higher nubers, but calculation will take super lo
ng)
NODE=1:50 # Minimum size of terminal nodes
MinErr=1
MinNT=0
minNDT=0
minMT=0

for(nt in NTrees){
  for(mt in MTRY){
    for(nd in NODE){
      ran_for2 = randomForest(stroke~., type="classification", train, ntree=nt, mtry=mt, node
size=nd, importance=TRUE)
      rf_predict2=predict(ran_for2, train)
      Err=mean(rf_predict2 != train$stroke)
      if(Err < MinErr){
        MinErr=Err
        minNT=nt
        minNDT=nd
        minMT=mt
        #print(c("NT=",nt," MT=",mt," NDT=",ndt," minE=",MinError))
      }
    }
  }
}
print(c("NTrees=",minNT," MTRY=",minMT," NODE=",minNDT," MinErr=",MinErr)) # best models vari
ables to minimize the error
```

```
## [1] "NTrees="             "7"                   " MTRY="
## [4] "6"                   " NODE="              "1"
## [7] " MinErr="            "0.00562622309197652"
```

The calculated variables are inserted into the model

```
ran_for3 = randomForest(stroke~., type="classification", train, ntree=minNT, mtry=minMT, node
size=minNDT, importance=TRUE)
summary(ran_for3)
```

```
##                 Length Class  Mode
## call               8   -none- call
## type               1   -none- character
## predicted       4088   factor numeric
## err.rate          21   -none- numeric
## confusion          6   -none- numeric
## votes           8176   matrix numeric
## oob.times       4088   -none- numeric
## classes            2   -none- character
## importance        40   -none- numeric
## importanceSD      30   -none- numeric
## localImportance    0   -none- NULL
## proximity          0   -none- NULL
## ntree              1   -none- numeric
## mtry               1   -none- numeric
## forest            14   -none- list
## y               4088   factor numeric
## test               0   -none- NULL
## inbag              0   -none- NULL
## terms              3   terms  call
```

```
ran_for3
```

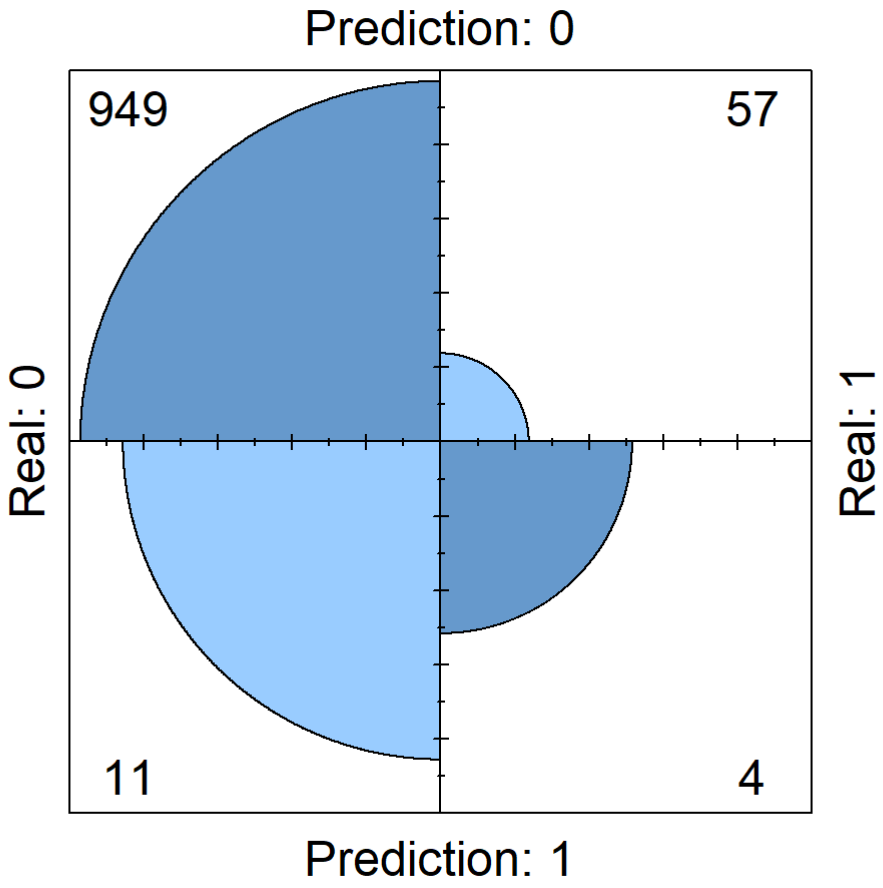
```
##
## Call:
##  randomForest(formula = stroke ~ ., data = train, type = "classification",      ntree = mi
nNT, mtry = minMT, nodesize = minNDT, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 7
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 6.87%
## Confusion matrix:
##      0   1 class.error
## 0 3640 108  0.02881537
## 1  162  21  0.88524590
```

Test the model using the test set

```
rf_predict3=predict(ran_for3, test)
```

Check prediction results

```
fourfoldplot(table(Prediction = rf_predict3, Real = test$stroke), conf.level = 0, margin = 1)
```




```
print(1-mean(rf_predict3 != test$stroke))
```

```
## [1] 0.9333986
```

With optimal model variables, the model is not performing much better The accuracy is still high But with still only few true positives the model is still not very useful

Oversampling

next, the model is tested with the oversampling data

```
table(data2$stroke) # qick look at oversampled data
```

```
##
##    0    1
## 4860 2104
```

The train and test dataset from oversampling is transformed into a factor

```
train2$stroke <- as.character(train2$stroke)
train2$stroke <- as.factor(train2$stroke)
test2$stroke <- as.character(test2$stroke)
test2$stroke <- as.factor(test2$stroke)
```

Create random forest model with oversampled data

```
ran_for4 = randomForest(stroke~., train2, importance=TRUE)
summary(ran_for4)
```

```
##                 Length Class  Mode
## call               4  -none- call
## type               1  -none- character
## predicted        5572  factor numeric
## err.rate         1500  -none- numeric
## confusion           6  -none- numeric
## votes           11144  matrix numeric
## oob.times        5572  -none- numeric
## classes             2  -none- character
## importance         40  -none- numeric
## importanceSD       30  -none- numeric
## localImportance     0  -none- NULL
## proximity           0  -none- NULL
## ntree               1  -none- numeric
## mtry                1  -none- numeric
## forest             14  -none- list
## y                5572  factor numeric
## test                0  -none- NULL
## inbag               0  -none- NULL
## terms               3  terms  call
```

```
ran_for4 # the error rate has fallen sharply
```

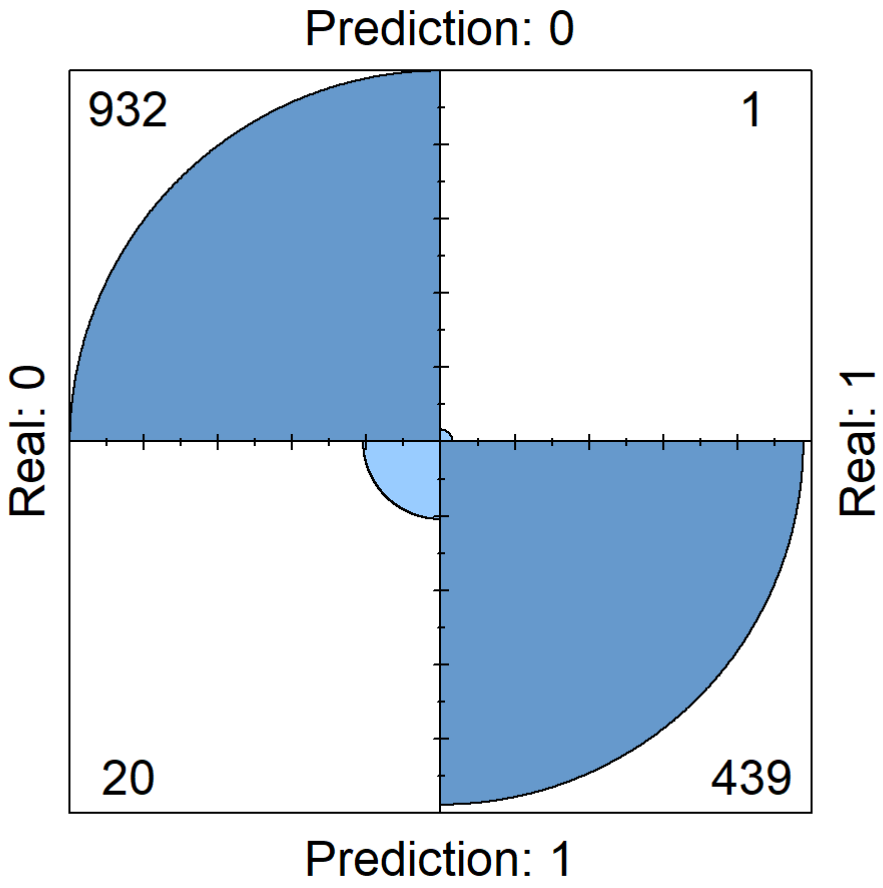
```
##
## Call:
##  randomForest(formula = stroke ~ ., data = train2, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 1.56%
## Confusion matrix:
##      0    1 class.error
## 0 3825   83 0.021238485
## 1    4 1660 0.002403846
```

Test model on test data

```
rf_predict4 = predict(ran_for4, test2)
```

Check prediction results

```
fourfoldplot(table(Prediction = rf_predict4, Real = test2$stroke), conf.level = 0, margin = 1
)
```




```
print(1-mean(rf_predict4 != test2$stroke))
```

```
## [1] 0.9849138
```

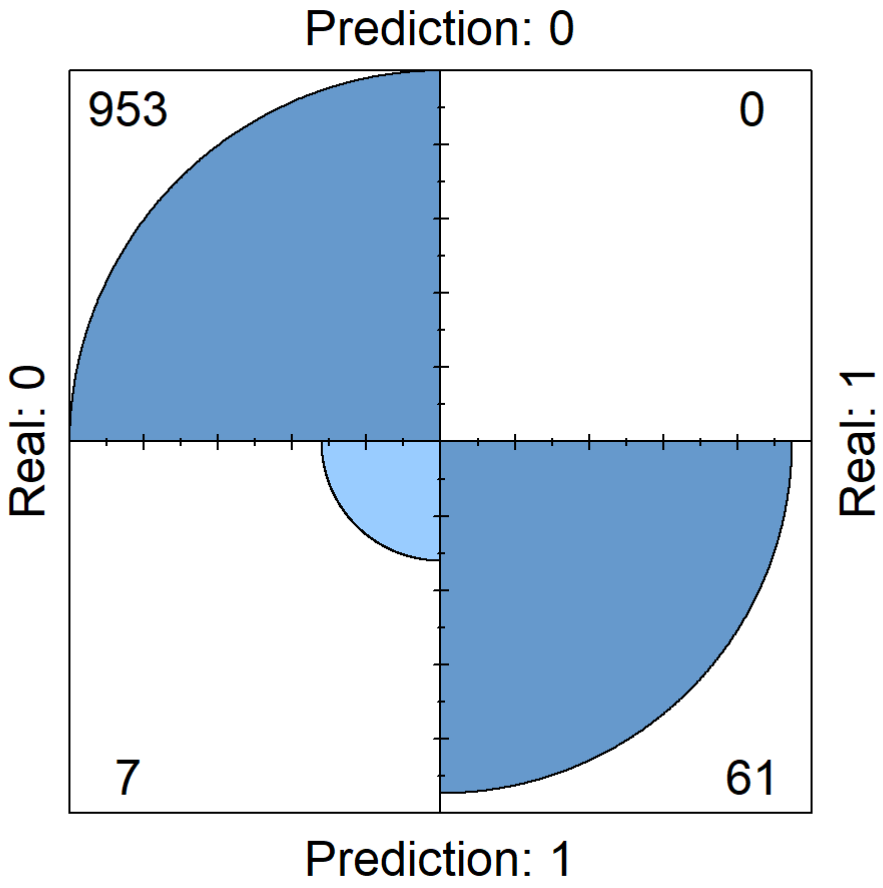
With oversampled data the model performed way better. The number of False positives and false negatives is quit low and the number of true positives is quite high The accuracy is also high

Test model on test data without oversampling

```
rf_predict5 = predict(ran_for4, test)
```

Check prediction results

```
fourfoldplot(table(Prediction = rf_predict5, Real = test$stroke), conf.level = 0, margin = 1)
```




```
print(1-mean(rf_predict5 != test$stroke))
```

```
## [1] 0.993144
```

Also with not oversampled test date the model which was trained with oversampled data performes quite good. The accuracy is also high and the number of true positives is also high. The number of false positves and fales negatives aus low. So far the random forest model, trained with oversampled data performed the best.

XGBoost

As last model XGBoost will be used since is very powerful for classification and regression Especially in many ML competioions XGBoost has achieved good results. Also XGBoost models are widely used machine learning algorithms nowadays.

Create train and test data

```
set.seed(5)
test_index <- createDataPartition(data$stroke, times = 1, p = 0.75, list = FALSE)
test3 <- data[-test_index, drop=FALSE]
train3 <- data[test_index, drop=FALSE]

dim(train3)
```

```
## [1] 3832   11
```

```
dim(test3)
```

```
## [1] 1277   11
```

Transform the searched variable, stroke, into a factor

```
train$stroke <- as.character(train$stroke)
train$stroke <- as.factor(train$stroke)
test$stroke <- as.character(test$stroke)
test$stroke <- as.factor(test$stroke)
```

Since it can be very complex to experiment with all settings, widely used settings were taken.

```
grid <- expand.grid(nrounds = 3500, max_depth = 7,eta = 0.01, gamma = 0.01, colsample_bytree
 = 0.75, min_child_weight = 0, subsample = 0.5) # create grid with standard values
Control <- trainControl(method = "cv", number = 5)
```

Train the model

```
xgb_model <- caret::train(stroke ~ ., train, method = "xgbTree", tuneLength = 3, tuneGrid = g
rid, trControl = Control)
```

```
xgb_model
```

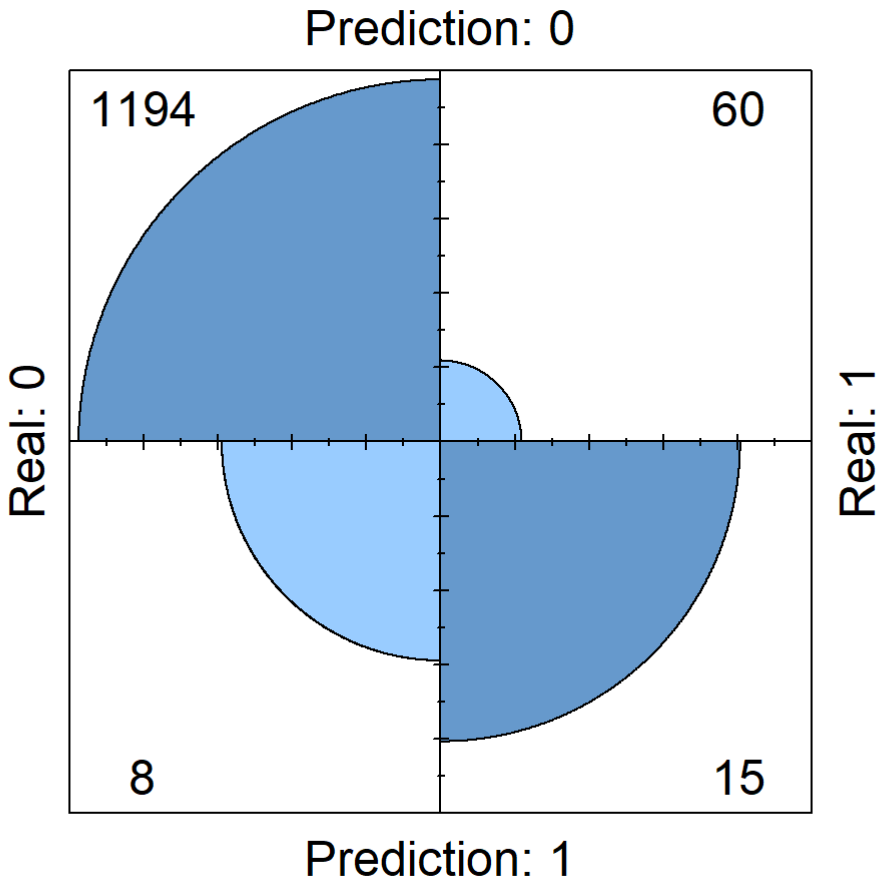
```
## eXtreme Gradient Boosting
##
## 4088 samples
##   10 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 3270, 3270, 3270, 3271, 3271
## Resampling results:
##
##   Accuracy   Kappa
##   0.9515653  0.07788143
##
## Tuning parameter 'nrounds' was held constant at a value of 3500
##
## Tuning parameter 'min_child_weight' was held constant at a value of 0
##
## Tuning parameter 'subsample' was held constant at a value of 0.5
```

Test model with test data

```
xbg_pred <- predict(xgb_model, newdata = test3)
```

Check prediction results

```
fourfoldplot(table(Prediction = xbg_pred, Real = test3$stroke), conf.level = 0, margin = 1)
```




```
print(1-mean(xbg_pred != test3$stroke))
```

```
## [1] 0.9467502
```

This model also delivers good results. The true positives are much highter than on the first model wile the accuracy is as high as for the first model.

Results

After an initial data review and data cleaning, three different ML algorithms were used to predict strokes. The first model is a logistic regression model. With the default settings, a high accuracy could be achieved, but the results were not particularly good, since there were only very few ture positives. In the first step, experiments were made with the threshold value above which a prediction is counted as a hit. The threshold value of 0.1 was finally taken. With this setting the number of false positives is the highest, but the number of false negatives is the lowest. For the data used, false positives are less bad than false negatives. For the fact that the number of true positives goes up, the number of false positives also goes up. In this case, however, it is better to predict false positives and have more true positives than to have fewer false positives and fewer true positives. At the expense of Accuracy the number of true positives increases With these results, the model could be used as a kind of warning system in this case. In the further course, experiments were carried out to reduce the influencing variables of the model. On the one hand, variables were selected manually on the basis of the previous data analysis, and on the other hand, an automated approach was used to reduce the number of variables. However, the results were very similar to the first results with this model. Since comparatively few data with stroke were available, the method of oversampling was tested in a final cut. With this approach the number of false negatives is lower compared to the other aproaches without oversampling, what is good from medical point of view. The accuracy droped significantly. If the model trained with the oversampled data is applied to the test data without oversampling the number of false negatives is very low and the number of true

positives is the highest of all those tested. From a medical point of view, this model is therefore the best at first glance. Unfortunately, the number of false positives is also very high and the accuracy very low. Even if the model could be used as an early warning system, a lot of warnings would be issued. So it looks like the first model is still the best so far.

Next, a Random Forest model was used. With the default settings, the results were similarly poor as with the first model with default settings. Subsequently, the model parameters were adjusted using an automated approach and the model was able to achieve slightly better results. But with optimal model variables, the model is still not performing much better. The accuracy is still high, but with still only few true positives the model is still not very useful. For this reason, oversampling was also applied in this model. With oversampled data the model performed way better. The number of False positives and false negatives is quit low and the number of true positives is quite high. The accuracy is also high. Also with not oversampled test date the model which was trained with oversampled data performes quite good. The accuracy is also high and the number of true positives is also high. The number of false positves and fales negatives is low. So far the random forest model, trained with oversampled data performed the best.

The last model used was an XGBoost model. The model was not tested in standard settings, since the previous models both performed quite poorly with standard settings. The model was therefore used with frequently used settings from other test cases. This model also delivers good results. The true positives are much highter than on the first model wile the accuracy is as high as for the first model.

In summary, the Random Forest model trained with oversampling data gave the best results. The decision whether the XGBoost model or the Logistic Regression model yields better results is difficult to make. The Logistic Regression model predicts many false positives, but also the most true positives and the fewest false negatives, which is important from a medical point of view. The XGBoost model predicts few false positives and relatively many true positives for the first attempt. But also many false negatives and since this is a disease that is usually fatal, it is important from a medical point of view to predict few false negatives and rather too many false positives.

From a medical point of view, therefore, the logistic regression model is probably the second-best.

Conclusion

Starting from a dataset containing clinical patient data, 3 different ML models, including advanced ML models, were applied to predict whether or not a stroke was present based on different factors. One problem with the data was that there were relatively few patients with stroke in the data. This problem was at least partially overcome by oversampling. Based on the analyses, as explained in the Results section, a random forest model was best suited for prediction. In general, this work can be used to better understand the factors leading to stroke and to predict strokes. With the help of such models, it may be possible to predict patients who are at increased risk of stroke due to various factors. As a next step, the individual model parameters could be further adjusted to improve the results. In addition, the models could be tested on even larger data sets and further clinical factors could be included in the data. So far, the model results have mainly been subjective compared from a medical point of view. The model results could also be compared on the basis of various KPIs to find the best scientific model.